

### 版权相关注意事项：

- 1、书籍版权归著者和出版社所有
- 2、本PDF来自于各个广泛的信息平台，经过整理而成
- 3、本PDF仅限用于非商业用途或者个人交流研究学习使用
- 4、本PDF获得者不得在互联网上以任何目的进行传播
- 5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
- 6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
- 7、请于下载PDF后24小时内研究使用并删掉本PDF

O'REILLY®

仅供非商业用途或交流学习使用



# Effective DevOps

(中文版)

---

Effective DevOps

Jennifer Davis, Ryn Daniels 著

刘海涛 肖斌 等译

中国电力出版社



## 对本书的赞誉

“只要有兴趣了解 devops 以及如何培养 devops 文化，  
就会发现这是一本绝妙的书。”

——James Turnbull, Kickstarter CTO

“devops 不是另一个可以推迟和废止的‘技术运动’；它的触角延伸到整个组织中从概念到现金的方方面面。任何人如果发现他们构建新特性的成本在增加，或者经历过意外宕机的窘迫，就会理解 devops 运动所倡导的内在价值。”

“本书是我看过有关这个主题的最全面的一本书，不论是技术线和管理线的人，都可以很好地理解这本书并加以应用。我特别喜欢揭露有关 devops 的常见故事和传言的那些部分，例如说 devops 能降低成本、只有创业公司可用，或者认为这只是一个职位。这些响亮的反驳声掩盖了真正的 devops 精神，妨碍了这个运动的普及。所有设计、开发和部署软件的组织都可以把这本书作为推荐读物。”

——Nivia S. Henry, Sr. Summa 敏捷教练

“高绩效的组织把技术看作一种战略能力。不过，他们理解技术通常不是最有挑战性的方面。文化才最有挑战性。你的组织是否能很好地协作、试验和努力学习和分享知识，这会反映出它的接受水平以及所要达到的水平。在这本书中，本书作者描述了创建一个实用系统的必要条件，使你的企业可以沿着自己的道路前进，实现持续的改进、成长和转变。仔细阅读这本书，然后反思，再着手部署。”

——Barry O'Reilly, ExecCamp 创始人和 CEO，

《Lean Enterprise: How High Performance Organizations Innovate At Scale》

(O'Reilly) 合作者

“通过仔细分析 devops 运动的历史和传统，  
为当前从业人员提供了详细的建议。”

——Bridget Kromhout, Pivotal 首席技术专家

“本书是一本全面优秀的技术合集，这些构成了出版《敏捷宣言》以来技术工作的最大变革。它集合了来自不同行业不同来源的故事和资源，以一种可实现的方式介绍所有内容，不仅介绍了工具，还讨论了文化、行为和工作模式特性，这些对于成功的技术团队是必不可少的。”

——Mandi Walls, Chef 技术社区经理和

《Building devops Culture》(O'Reilly) 作者

“每个人都知道文化在 devops 环境中的重要性，但是对于这个重要的主题，一直以来都没有一整本书进行研究。本书对人为因素做了广泛、深入的调查，每一个希望建立高绩效技术团队和组织的管理者都应当仔细研究这些因素。”

——Jez Humble, 《Continuous Delivery》(Addison-Wesley)

和《Lean Enterprise》(O'Reilly) 的合作者



# Effective DevOps

(中文版)

Jennifer Davis & Ryn Daniels 著

刘海涛 肖斌 等译

Beijing • Boston • Farnham • Sebastopol • Tokyo

**O'REILLY®**

O'Reilly Media, Inc. 授权中国电力出版社出版

中国电力出版社

Copyright © 2016 Jennifer Davis and Ryn Daniels. All rights reserved.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and China Electric Power Press, 2018.  
Authorized translation of the English edition, 2016 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版 2016。

简体中文版由中国电力出版社出版 2018。英文原版的翻译得到 O'Reilly Media, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

## 图书在版编目 (CIP) 数据

Effective DevOps / (美) 珍妮佛·戴维斯(Jennifer Davis), (美) 莱恩·丹尼尔斯(Ryn Daniels) 著; 刘海涛等译. —北京: 中国电力出版社, 2018.2

书名原文: Effective DevOps

ISBN 978-7-5198-1419-9

I. ①E… II. ①珍… ②莱… ③刘… III. ①软件工程 IV. ①TP311.5

中国版本图书馆CIP数据核字(2017)第293103号

北京市版权局著作权合同登记 图字: 01-2017-7621号

---

出版发行: 中国电力出版社

地 址: 北京市东城区北京站西街19号 (邮政编码100005)

网 址: <http://www.cepp.sgcc.com.cn>

责任编辑: 刘 焱 (liuchi1030@163.com)

责任校对: 太兴华

装帧设计: Randy Comer, 张 健

责任印制: 蔺义舟

---

印 刷: 三河市百盛印装有限公司

版 次: 2018年2月第一版

印 次: 2018年2月北京第一次印刷

开 本: 787毫米×980毫米 16开本

印 张: 22.25

字 数: 425千字

印 数: 0001—3000册

定 价: 88.00元

---

版 权 专 有 侵 权 必 究

本书如有印装质量问题, 我社发行部负责退换



# O'Reilly Media, Inc. 介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

## 业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

# 目录

序一 .....	1
序二 .....	3
前言 .....	5

## 第一部分 什么是DevOps

第1章 devops概貌 .....	17
devops文化快照 .....	17
文化演进 .....	19
这个故事的意义 .....	21
Ryn的故事 .....	22
Jennifer的故事 .....	23
通过故事描述devops .....	25
第2章 什么是DevOps .....	26
文化处方 .....	26
DevOps是多方面的综合 .....	26
第3章 DevOps的历史 .....	31
开发人员作为运维人员 .....	31
软件工程的出现 .....	32



专有软件和标准化的出现 .....	33
网络时代 .....	34
全球社区的起源 .....	35
应用和Web时代 .....	36
软件开发方法论的发展 .....	37
开源软件，专有服务 .....	38
敏捷基础设施 .....	38
devopsdays的起源 .....	40
devops的现状 .....	40
小结 .....	41
<b>第4章 基本术语与概念 .....</b>	<b>42</b>
软件开发方法论 .....	42
运维方法论 .....	45
系统方法论 .....	46
开发、发布和部署概念 .....	48
基础设施概念 .....	50
文化概念 .....	53
小结 .....	55
<b>第5章 DevOps误区和反模式 .....</b>	<b>56</b>
常见的devops误区 .....	56
devops反模式 .....	64
小结 .....	66
<b>第6章 有效实现devops的4大支柱 .....</b>	<b>67</b>
协作 .....	68
亲密性 .....	68
工具 .....	68
规模化 .....	68
小结 .....	69

亲密性需求 .....	154
度量亲密性 .....	157
Sparkle公司开发和运维的亲密性 .....	160
小结 .....	160
<b>第10章 亲密性：误区和问题排查 .....</b>	<b>161</b>
亲密性误区 .....	161
亲密性问题排查 .....	162
<b>第四部分 工具</b>	
<b>第11章 工具：生态系统概览 .....</b>	<b>173</b>
软件开发 .....	173
自动化 .....	177
监控 .....	182
生态系统的演进 .....	186
小结 .....	187
<b>第12章 工具：文化加速器 .....</b>	<b>188</b>
工具对人的意义 .....	188
工具是什么？ .....	189
解决实际问题的适当工具 .....	189
采用开源 .....	190
工具标准化 .....	191
工具分析的一致流程 .....	192
标准化的例外情况 .....	192
工具不重要 .....	193
工具对文化的影响 .....	194
工具的选择 .....	197
审查你的工具生态系统 .....	201
工具消除 .....	201
案例研究 .....	203



分析DramaFever.....	203
DramaFever的工具选择.....	208
分析Etsy.....	210
动因和决策挑战.....	218
Sparkle公司有效地使用工具.....	218
小结.....	219
<b>第13章 工具：误区和问题排查.....</b>	<b>221</b>
工具误区.....	221
工具问题排查.....	224

## 第五部分 规模化

<b>第14章 规模化：拐点.....</b>	<b>229</b>
理解规模化.....	229
考虑企业devops.....	230
组织结构.....	232
团队灵活性.....	233
组织生命周期.....	234
复杂性和变更.....	237
案例研究：团队发展和规模化.....	249
团队规模化和成长策略.....	257
组织规模化.....	267
案例研究：政府数字服务.....	269
案例研究：Target.....	275
分析Target.....	275
小结.....	280
<b>第15章 规模化：误区和问题排查.....</b>	<b>282</b>
规模化误区.....	282
规模化问题排查.....	285
我们不知道是否需要为X建立一个完整的团队.....	289

## 第六部分 搭建DevOps文化桥梁

### 第16章 利用有效实现DevOps的4大支柱搭建桥梁 ..... 293

故事的意义 .....	294
理论和实践中的devops .....	296
小结 .....	298

### 第17章 搭建DevOps文化桥梁：从故事学习 ..... 299

什么故事可以让我们了解文化 .....	299
组织间的交互 .....	309
鼓励组织间的亲密性 .....	314
小结 .....	316

### 第18章 搭建DevOps文化桥梁：发展人际联系 ..... 317

关于工作的个人故事 .....	317
文化负债 .....	324
系统的健康 .....	325
小结 .....	331

### 第19章 结论 ..... 332

接下来做什么 .....	333
创建有效的DevOps .....	334

### 第20章 更多资源 ..... 336

什么是DevOps? .....	336
协作：个人合作 .....	337
亲密性：从个人到团队 .....	338
工具：文化加速器 .....	338
规模化：拐点 .....	339
搭建DevOps文化桥梁 .....	341
推荐会议和线下聚会 .....	341
推荐播客 .....	342



# 序一

软件开发和运维领域正在发生巨变，这不仅仅是在我们的词典中再引入一个新词，绝非如此。在这个世界里，有关软件的设计、构建和运维的观念发生了根本性的转变，几乎每一个成功的组织都认识到软件不只是可以构建和上线，它还需要运维。

这个转变的独特之处在于，它涵盖面更广，更强调整体，而且更能反映工程团队日常面对的现实。原先制造和装配线领域的概念可以直接用在软件开发和运维中，这样的日子已经一去不复返了。之前，产品可以经过设计、规划然后最终上线，这样的日子也远去了。不再有“最终”，只有一个不断调整、改变和学习的无限循环。

团队和组织中的工程师想让他们的工作变得“简单”，这会带来大量复杂性，本书作者在这本书中为处理这些复杂性给出诸多可以参考的思路。

本书作者没有描绘一个完美的画面或确定的解决办法。实际上，她们描述了一些团队和组织的不同主题领域、实践和观察结果，这些团队和组织充分理解这样一种思想：好的产品、好的用户体验以及好的软件，其核心是人的合作、全面的评论、有效的协作以及判断的完美统一。

2009年，我的朋友Paul Hammond与我在O'Reilly的Velocity大会发表了题为“10+ Deploys a Day: Dev and Ops Cooperation at Flickr”（每天部署10次以上：Flickr公司的Dev与Ops的合作）的演讲。尽管其中一些内容是关于持续部署的想法，但是很多人都选择关注“10次以上部署”部分而不是“合作”部分。在我看来，如果认为技术或“硬部分”可以与社会和文化“软部分”分开和脱离，这种看法是一个错误。并不是这样，它们不能分开。它们是紧密关联在一起的，对成功同等重要。换句话说，人和过程对工具和软件有很大影响，尽管大多数人不愿意承认或者甚至不知道这一点。

我强烈建议你：首先，也是最重要的，不要犯这种错误，不要认为技术与人和过程无关。一旦你开始这样想，你就输了。

这些主题在通常的计算机科学学科中是找不到的，在一般的专业领导和开发课程中也没有。这些内容来自具体实践，只有通过这个领域中的艰苦工作才能得来。

本书作者在这本书中为你提供了很全面的一组路标。对于你，亲爱的朋友，衷心希望你在自己的上下文和环境仔细考虑这些路标。

——John Allspaw  
Etsy首席技术官  
Brooklyn, NY

## 序二

2003年，Nicholas Carr向世界宣布IT不再重要。因为他是在“哈佛商业评论”上这么说，所以很多组织（以及组织的领导人）都相信他。嗯，时代已经改变，IT也是一样。从2009年开始，最具创新力的团队和组织已经表明，技术在提供真正的价值和竞争优势方面可以扮演一个关键的角色。这种技术革新称为DevOps，这本书会向你展示如何加入这些创新公司，利用技术提供价值。

在这本书中，本书作者利用她们在创新公司的丰富经验以及作为社区卓越专家的背景，重点介绍了有效地实现DevOps（或者她们所称的devops）到底要做什么。基于这种优势，她们提供了对所有读者都适用而且有益的独特见解，因为她们综合了多家公司以及多个行业的知识。对于读者，不论你正处在devops旅程的哪个阶段，也不论你的组织规模大小，都能从中获益。

本书给出的故事和建议完全类似于过去十年里我自己的工作中见到的情况。作为这个领域的首席研究员和“DevOps现状报告”的首席调查员，我知道，强调信息流动和信任的强大组织文化是所有DevOps变革的关键要素，这也是DevOps运动区别于传统IT的重要因素。我从超过20000名DevOps专业人士收集的数据也表明，这种文化会提升IT和组织绩效，使最好的IT组织相比于其他组织可以得到双倍的生产力、利润和市场份额。本书作者在书中首先讨论了文化、沟通和信任等方面，并用大量时间讨论这些因素对所有变革工作的重要性，这并不是失误。作为技术人员，我们总喜欢从工具甚至过程开始讨论，但是实际数据反复表明，除了前面提到的IT和组织绩效，文化对于工具和技术成功至关重要。第二部分到第三部分将分别讨论协作和亲密性，不论你是刚开始你的DevOps变革，想知道要实现什么和注意什么，还是要让你现有的



DevOps实践更上一层楼，寻求方法来实现优化和排查问题，这两部分都是必读的。

作为一些最具创新力的公司的顾问，我发现实现DevOps和规划一个技术变革路线图时，最具挑战性的部分是帮助团队和组织理解并不存在一个正确答案，是否正确总是取决于你的团队和你的组织。我很高兴Ryn和Jennifer在这本书中很好地体现了这一点，指出不可能“即插即用”地实现一个解决方案，并介绍了成功构建你自己的DevOps解决方案所需的工具和技术。除了第二到第三部分，还应当仔细阅读关于工具的第四部分，这对于所有DevOps变革都是必不可少的。我尤其欣赏她们不仅谈到了技术，还描述了所在文化的关键要素。

这本书中我最推崇的一点是各种不同的读者都可以理解。关于规模化的第五部分对于个人贡献者和团队领导尤其有用，我不仅自己使用这一部分作为参考，对我的客户也会参考这一部分。第4章介绍术语，第11章概要介绍生态系统，这对于技术人员（确保我们达成共识，因为不同的团体通常都使用不同的说法）和可能使用一个现用参考的领导都很方便。对于还没有经过这种训练的大学生，这本书可以作为这个领域的一个必读的入门介绍，我真希望我做教授的时候这本书已经问世。

我们生活和工作在这样一个激动人心的时代，技术已经集成到我们的核心业务中，这使得每个公司都成为了一个软件公司。现在技术提供了新的机会，可以采用全新的方式以此前不可能的速度向客户交付特性。组织往往需要努力跟上。老的IT和瀑布方法无法保证组织足够快地传递价值，这种情况我在数据中见过，另外我还为一些客户和公司创建解决方案来实现他们的DevOps旅程，在这些客户和公司里我也见过这种情况。本书作者看到了采用老的方式实现技术变革遇到的挑战，也看到了利用DevOps可以提供的绝好机会，因此她们写了这本书，来引导我们完成自己的旅程。好好读这本书，开始你自己的冒险！迭代、学习、成长，然后再次开始你自己的冒险！

——Nicole Forsgren博士  
主管，Chef Software  
华盛顿西雅图

# 前言

想象这样一种场景：一个小型Web公司遇到了问题，由于前所未有的飞速成长，公司的网站开始难以招架，经常出故障。员工需要花更多的时间维护服务，同时还要实现和部署新特性，对此大家越来越不满。由于存在语言和时区障碍，也让分布在全球各地的团队之间摩擦增加。网站崩溃时的紧张反应滋生了一种问责文化，导致团队之间出现更多的猜疑，同时透明度降低。

针对这些问题，这个组织认为看起来devops像是一个不错的解决方案。管理层聘用了很多新人加入他们的新devops团队。这个devops团队的职责是待命，现在的运维团队可以打电话给他们，来解决他们不知道如何处理的问题。devops团队成员比运维团中的人有更多行业经验，所以他们通常有更多的知识储备来处理生产问题。不过，由于没有时间或机会来学习新技能，运维人员会反复提出同样的问题。

devops团队厌倦了作为开发和运维团队之间的中间人。管理层的“解决方案”并没有缓解问责文化，而是导致沟通加倍不力，因为任何一个团队都对计划过程、email、聊天消息，甚至其他团队的bug跟踪毫不知情。

因此，管理层宣布“这个devops方案”失败，不愿意再投入更多时间、精力或资金给运维或devops团队，认为他们工作不得力，总是“让网站关闭”，“妨碍”了“真正的”开发工作的顺利完成。这两个团队里能找到更好工作的成员都离开这家公司，投奔了其他组织（在那些组织中，大喊大叫和相互责备是不可接受的），这就导致剩下的团队效率更低。



# 引入有效的DevOps

哪里出了问题？看上去devops是一个很好的想法，不过创建devops团队却带来让人失望的结果。如果采用其他方式，这个组织要怎样做才能真正改善他们的情况，得到针对问题的具体解决方案？在这本书中，我们会介绍如何采用一种devops思维模式实现有效的改变。

这本书并不是一个处方，不会告诉你实现devops的“唯一正确的道路”。我们不会为你提供一个成品devops或者“devops作为服务”，也不会说你的DevOps做法是错的。这本书会介绍一组概念和方法来改善个人协作、团队和组织亲密性以及整个公司或组织中的工具使用，并解释这些概念如何支持组织根据需求来改变和调整。每个组织都是独一无二的，尽管没有十全十美地实现devops的方法，但是对于每一个希望提高其软件质量同时又改善员工效率和福利的组织，可以采用不同的方式应用这些共同主题。

效率是把事情做对。有效是做对的事情。

——Peter F. Drucker

有效定义为做对的事情，并得到期望的结果。要做对的事情，我们必须理解我们的目标，并了解具体的短期目标如何服务于这些目标。

我们希望能帮助你根据当前文化明确你的环境中哪些是对的事情，包括你正在使用的过程和工具。我们在这本书中分享的原则和见解可以在你的整个组织里应用，而不仅限于开发和运维团队。我们甚至发现，在写这本书的过程中自己也在应用这些原则。

尽管我们写作本书总的目标是分享共同的故事、技巧和实践，但每个组织都可以采用和应用他们自己的工作方法，我们都有自己的故事和经验。从私营到公共部门，从小的创业公司到大型企业，以及从开发到运维、质量保证、咨询等各种角色，我们从不同方面搜集了经验，这些经验汇集在一起为我们提供了丰富的信息，在写这本书的过程中它们是很好的补充。

## “devops”、“devops”还是“DevOps”？

对于“devops”这个词的字母是否大写我们有过很多讨论。一个简单的在线调查显示，大多数人都支持“DevOps”。我们还发现组织比较关注“Dev”和“Ops”，以至于出现了“DevSecOps”和“DevQAOps”，因为“DevOps”隐舍地只包括“Dev”和“Ops”。



所以最后我们选择了“devops”，这对应原来推特上的标签，用来联系希望改善协作的人，他们希望谈话时不再区分“你们”和“我们”，而应当更多地讨论如何利用可持续的工作实践（强调人为因素）来支持企业发展。

成功的项目需要有适当的输入、努力、见解以及整个组织中人们之间的协作，你的组织中存在的问题可能不只限于开发人员和运维团队。我们在这本书中有意选择使用小写的“devops”来反映我们的观点，这只是一种包容，并没有其他的意思。

## 本书面向的读者

一些有领导角色的管理者和个人贡献者可能看到了他们的组织中存在的摩擦，希望找到并采用具体可操作的步骤，在他们的工作环境中实现或改善devops文化，他们就是本书面向的读者。不仅如此，各个层次的个人贡献者如果希望得到实用的建议来缓解痛点问题，都能从这本书中有所收获。

我们的读者有各种不同的专业角色，因为devops是一个专业和文化运动，强调迭代的工作，从而打破信息孤岛、监督关系，以及修正组织中团队之间产生的误解。

这本书涵盖大量devops技能和理论，包括基本思想和概念的介绍。我们假定你听说过“devops”这个词，对这个领域使用的工具和过程有基本的了解。

建议你先把严格的定义放在一边，对于我们发现的最有效的devops原则，应当保持一种开放的心态。

读完这本书之后，你会深入地了解devops文化对你的组织意味着什么；如何鼓励有效的协作，从而使有不同背景的个人贡献者和有不同目标和工作方式的团队能高效率地协同工作；如何帮助团队协作，从而最大限度地体现他们的价值，同时提高员工满意度并平衡冲突的组织目标；如何选择工具和工作流程来完善你的组织。

## 本书如何组织

我们花了很长时间来考虑这本书中章节的顺序和组织。由于并不存在实现devops的唯一正确的方法，所以对于“如何实现devops”也没唯一正确的顺序。读这本书的每个人可能处在devops旅程的不同阶段，而且每个旅程都有不同的故事，有不同的道路，而且需要解决的问题和冲突也不同。

这本书分为多个部分（第一部分，我们会综观全局，为你介绍devops的概貌，然后更细致地介绍devops思想、定义和原则；第二部分到第五部分介绍有效实现devops的4大支柱；第六部分是最后一部分，会讨论如何使用故事在个人、团队和组织之间建立联系）：

- 第一部分，“什么是DevOps”。
- 第二部分，“协作”。
- 第三部分，“亲密性”。
- 第四部分，“工具”。
- 第五部分，“规模化”。
- 第六部分，“搭建DevOps文化桥梁”。

第二部分到第五部分将分别讨论有效实现devops的一个支柱，这些部分的最后分别有一章专门讨论有关这个支柱的各种误区，并介绍关于该主题的常见问题排查方案。如果你要在自己的组织中实现devops，倘若在这些领域遇到麻烦，会发现这些“误区和问题排查”给出了更实用的建议来指导他们解决问题。

如果你认为使用计算机比用人来完成工作更容易，可能想跳过本书中关于人际关系和文化的内容，不过这些方面会让我们对如何合作有所认识，这包括文化和技术如何交互，而文化和技术的结合正是devops的优势所在。

在你的故事中你会有自己的位置，可以根据你所在的位置选择最合适的顺序阅读这本书中的章节；可以把它看作是一本“可以自由冒险”的书。4大支柱相互交织、相互关联，希望你以后还能再回来重读对你们有意义的部分，并继续在你们的devops旅程中学习这些原则。

## 案例研究方法

在这本书中，我们分享了这个行业很多公司中个人的故事。这些信息是通过采访组织中不同层次的人、发表的博客文章、演示文稿以及公司文件收集的。尽管每一章的主题指出了相关案例研究的方向（即某个支柱），不过由于devops的特殊性质，这意味着每个案例研究可能涉及多个支柱（甚至覆盖所有4个支柱）。

另外，除了提供比较正式的案例研究，我们还分享了非正式的故事和我们自己的个人经验，这些组合在一起展示了devops会以多种不同方式影响决策和个人故事。

这本书的目的就是要帮助你完成工作。一般来讲，如果本书提供了示例代码，你完全可以在你的程序和文档中使用这些代码，不需要联系我们来得到许可，除非你直接复制了大部分的代码。例如，如果你在编写一个程序，使用了本书中的多段代码，这并不需要得到许可。但是出售或发行O'Reilly示例代码光盘则需要得到许可。回答问题时如果引用了这本书的文字和示例代码，这不需要得到许可。但是如果你的产品的文档借用了本书中的大量示例代码，则需要得到许可。

我们希望但不严格要求标明引用出处。引用信息通常包括书名、作者、出版商和ISBN。例如“Effective DevOps by Jennifer Davis and Ryn Daniels (O'Reilly). Copyright 2016 Jennifer Davis and Ryn Daniels, 978-1-491-92630-7。”。

如果你认为在使用代码示例时超出了合理使用范围或者上述许可范围，可以随时联系我们：[permissions@oreilly.com](mailto:permissions@oreilly.com)。

## Safari®图书在线

Safari图书在线 ([www.safaribooksonline.com](http://www.safaribooksonline.com)) 是一个应需而变的数字图书馆，通过图书和视频方式提供世界顶尖作者在技术和商业领域积累的专家经验。

技术专家、软件开发人员、Web设计人员和企业以及有创意的专业人员都使用Safari图书在线作为其主要资源来完成研究、解决问题、深入学习和资质培训。

Safari图书在线为机构、政府部门和个人提供了多种产品组合和定价程序。

订阅者可以在一个快捷搜索的数据库中访问多家出版社提供的成千上万种图书、培训视频和正式出版前手稿，如O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology以及其他数百家出版公司。关于Safari图书在线的更多信息，请访问我们的在线网站。

## 联系我们

请将关于本书的意见和问题通过以下地址提供给出版商：



美国：

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）

奥莱利技术咨询（北京）有限公司

针对这本书，我们还建有一个网页，列出了有关勘误、示例和其他信息。可以通过以下地址访问这个页面：<http://bit.ly/orm-effective-devops>。

如果对这本书有什么意见，或者要询问技术上的问题，请将电子邮件发至：  
[bookquestions@oreilly.com](mailto:bookquestions@oreilly.com)。

## 致谢

如果没有众多朋友、同事和家人的帮助和指导，这本书不可能问世。我们要感谢O'Reilly的整个团队，特别要感谢Courtney Nash鼓励我们写这本书；感谢我们的编辑Brian Anderson，感谢他的所有支持；感谢帮我们选封面动物的秘密人物，谢谢他选择了长毛牦牛；另外还要感谢参与这本书的所有其他人，是你们让这本书得以出版。我们还要感谢Etsy的John Allspaw、Lara Hogan和Jon Cowie；Chef的Nicole Forsgren和Yvonne Lam；Pivotal的Bridget Kromhout；Stack Exchange的Tom Limoncelli，感谢他们的大力帮助、支持和一直以来的鼓励。

感谢参与案例研究的各位：Alex Nobert、Bridget Kromhout、Tim Gross、Tina Donbeck和Phaedra Marshall。

感谢所有分享故事的人，包括Davida Marion、Linda Laubenheimer、Hollie Kay、Nicole Johnson和Alice Goldfuss。

感谢我们的技术审校帮我们打磨文稿：Alice Goldfuss、Dustin Collins、Ernest Mueller、Matthew Skelton、Olivier Jacques、Bridget Kromhout、Yvonne Lam和Peter Nealon。

感谢Andy Paroff提供了Ed，就是这本书网站和贴纸上的那只可爱的devops牦牛。

## Ryn的致谢

感谢Etsy为我提供了机会来写这本书，允许我在那么多会议上演讲，还提供了那么完美的工作环境。尤其要感谢Web运维团队在这个项目过程中给予的支持和耐心，与你们的共事让我想起当初为什么喜爱这个工作。特别感谢Mike Rembetsy在我每次自认为不够好而拒绝面试时没有放弃我，感谢John Allspaw对我的鼓励和信任，感谢Laurie Denness和Jon Cowie的所有支持和渊博的知识，是你们帮助我这么快成长为一个运维工程师。

感谢Lara Hogan、Bridget Kromhout、Cate Huston和Melissa Santos，你们是最好的朋友和榜样，你们是能力超群的女性。认识你们和与你们交谈帮助我渡过了难关，你们的反馈和支持极大地帮助了我。

感谢James Turnbull多年前在推特上联系我，把我带入运维社区。衷心感谢认识你；感谢写书过程中你的支持、睿智和鼓励；感谢你让我成为一名铁杆的运维成员。

感谢Jason Dixon第一个邀请我在会议上发表演讲，相信我有值得一说的东西，尽管那时我都不相信自己。

感谢整个运维和devops社区，特别要感谢NYC运维人员提供的支持和新机会，还让我结识了很多朋友，可以和他们一起享受Sysdrink啤酒。

感谢Jennifer Davis，你是一个非常棒的朋友、会议伙伴和合作者。与你一起头脑风暴、写书、演讲、培训和编辑的过程真的很棒，衷心感谢能与你合作，另外要感谢我们庆祝时的所有小蛋糕，不论我们分处两地还是聚在一起。

最后，感谢我的妈妈支持、鼓励和信任我，我告诉过你即使有一头颜色怪异的头发我也能找到一份好工作！另外，如果没有我的猫在整个项目中不变的爱和支持，没有你暖腿，没有你趴在扶椅上陪我，这本书不可能问世。

## Jennifer的致谢

感谢Chef创造的工作机会，让我可以从众多不同的组织学习，并支持我通过演讲和培训分享我学到的知识。

感谢这个行业中的所有女性，你们引入了新的观点和可接受的行为和规范，这些改变了我们的工作方式，你们的意见真的很重要。请继续分享你们的经验，你们会得到大家的支持。

感谢整个devops社区，激发我希望成为这个社区的一部分。这个社区提供了一个支持系统，并鼓励可持续的工作实践。感谢所有分享个人故事和经验的人。

感谢Yvonne Lam、Bridget Kromhout、Dominica DeGrandis、Mary Grace Thengvall、Amye Scavarda、Nicole Forsgren和Sheri Elgin的深厚友谊。你们的想法和反馈帮助我形成了自己的观点。你们的支持对我是极大的鼓舞。

感谢Ryn Daniels，我的朋友和合作者，感谢我们一同完成这个项目过程中的种种合作，我们一起思考、撰写、编辑，以及在实践中示范我们的devops想法。有大笑和泪水，也有争吵，很荣幸能与你一起研究devops。

感谢我的祖母，小学老师Frances Wadsworth Hayes，是她启发我分享我的故事，一生都要学习。如果没有我的家人Brian Brennan和George的爱和支持，我什么都做不了。

## 第一部分

---

# 什么是DevOps



## 第1章

## devops概貌

devops是一种思维方式，同时也是一种工作方式。作为一个框架，它支持人们和团队分享经验、建立同理心，以有效而持久的方式实战使用。它也是文化组成的一部分，反映了我们要如何工作以及为什么这样做。很多人认为devops就是Chef或Docker之类的特定工具，不过这些工具本身并不是devops。工具之所以被认为是“devops”，关键在于它们的使用方式，而不是工具本身的基本特性。

除了用来实战使用的工具，我们的文化中还有一个同样重要的部分，这就是我们的价值观、标准和知识。通过分析人们如何工作、所使用的技术、技术如何影响我们的工作方式及人如何影响技术，将有助于明确我们的组织以及行业的格局，做出有意义的决策。

devops不单纯是一个软件开发方法论。尽管它与敏捷方法（Agile）或极限编程（XP）等软件开发方法论有关，甚至受到它们的影响，而且devops实践确实也包括软件开发方法或基础设施自动化和持续交付等特性，不过devops并不只是这些部分的简单累加。虽然这些概念确实与之相关，而且在devops环境中会经常看到，但是如果只关注这些部分就会一叶障目，而无法看到全貌，将忽视真正赋予devops强大功能的文化和人际关系等方面。

## devops文化快照

成功的devops文化是怎样的？为了说明这一点，我们将分析Etsy的人员、过程和工具的相互融合（Etsy是一个销售手工艺品和古董的全球在线集市）。我们之所以选择Etsy为例，不只是因为它们的技术和文化实践在业界很知名，此外还有一个原因：

Ryn（本书作者之一）在Etsy的工作经历可以为我们提供第一手资源，使我们能够从内部更详细地了解devps文化。

在Etsy，新入职的工程师第一天会拿到一个笔记本电脑，开发虚拟机已经建立了有适当访问权限和授权的帐户，而且已将最常用的GitHub仓库克隆到本地，还预建了重要工具的别名和快捷键，另外电脑桌面上有一个指南，其中包括新员工的信息以及公司其他资源的链接。由于不同团队使用的工具和实践得到了标准化，所以不论新人加入哪一个团队，都能更容易地快速进入状态，不过每个团队也有一定的灵活性，在适当情况下可以有所调整。

Etsy会安排一个老员工与这个新员工结对，老员工要带着新员工熟悉日常工作中会使用哪些测试和开发过程。首先老员工在他的本地开发虚拟机上编写代码，他的虚拟机的配置管理与实际生产环境的配置几乎是相同的。开发虚拟机设置为在本地运行和测试代码，所以在这些早期阶段，他能快速地运行和修改他的代码，而不会影响其他人的开发工作。

通过运行一组本地单元测试和功能测试，Etsy工程师可以很有信心地认为他们做的修改在本地能正常工作。在此基础上，他们会在try server上测试所做的修改，这是一个Jenkins集群，与公司的生产持续集成（continuous integration，CI）集群几乎完全相同，不过还有一个额外的优点，不需要向主干提交任何代码。如果通过了try测试，工程师就能更有信心地确信不会因为他们所做的修改导致任何测试失败。

取决于修改的规模和复杂性，新来的工程师可能选择新建一个拉取请求（pull request）或者不那么正式地请某个同事做代码审查。不是每一个修改都要求必须这么做，通常这由个人自行决定，Etsy提供了一个高度信任的无问责环境，在这个环境中，人们得到了充分的信任和授权，可以决定是否有必要做代码审查。会为新员工或初级员工提供指导，帮助他们确定哪些修改需要代码审查，以及谁来完成代码审查。作为一个新员工，在具体部署所做的修改之前，可以先让团队伙伴做粗略的检查。

如果已经通过本地测试和try测试，这个开发人员会加入Etsy所称的推送队列（push queue），把他的修改部署到生产环境。多个开发人员同时推送修改时，这个队列系统使用互联网中继聊天（Internet Relay Chat，IRC）和一个IRC机器人（IRC bot）协调部署工作。轮到这个工程师时，他提交的修改将推送到所用仓库的主干，并使用Deployinator将这些修改部署到QA。这会在QA服务器上自动开始构建，并运行整个CI测试套件。



成功地完成构建和测试之后，这个开发人员对网站的QA版本及日志做一个快速的人工检查，查找自动测试没能捕捉到的问题。之后，再使用同样的Deployinator过程将代码部署到生产环境，并确保测试和日志都一切正常。如果确实存在测试未能捕获的问题，可以利用大量图表仪表盘来监视，而且Nagios监控检查会提醒人们出现了问题。另外，很多团队有自己的轮班Nagios检查，鼓励所有人共同承担责任，保证一切平稳运行。如果未能平稳运行，人们会共同协作来帮助解决问题，无问责事后分析意味着人们可以从错误中学习，而不会因为犯错受到惩罚。

这个过程相当高效，从开始到完成平均只需要大约10分钟的时间，Etsy工程师们每天总共部署约60次。任何人都能得到他想要仔细研读的文档，在一个团队成员的指导下，每个工程师可以迅速熟悉这个过程，第一天就能向生产环境推送代码。即使是其他员工而非工程师，也通过“第一个推送项目”（First Push Program）鼓励其参与，他们可以与工程师结对来部署一个很小的修改，如把他们的照片增加到网站的员工页面。除了用于常规的软件开发，try测试和Deployinator过程非常好用，可以用于几乎任何可部署的项目，从开发人员用来构建虚拟机的工具到搜索日志的Kibana仪表盘，从Nagios监控检查到Deployinator工具本身，都可以采用这个过程。

## 文化演进

尽管Etsy现在是这样的，但几年前情况则完全不同，那时Etsy采用的部署过程还不太透明，而且很容易出错，部署时间长达约4小时。开发人员在自己的刀片服务器上工作，而不是使用虚拟机，不过那时刀片服务器的功能不够强大，还不能完整地运行自动测试套件。在过渡环境中运行的测试需要几小时才能完成，而且这些测试很不稳定，所以测试结果不是太有用。

工程组织中的团队就像一个个孤岛，相互割据。开发与运维之间存在“隐喻墙”，大量开发人员把代码扔给墙对面的运维人员，而运维人员只负责部署和监视，所以往往很抗拒改变。开发人员编写代码，执行手写的shell脚本来创建一个新SVN分支，然后使用svn merge部署（这不算是最容易使用的合并工具），将所有开发人员的修改合并到这个部署分支。开发人员要告诉运维工程师谁有权限部署哪一个要使用的分支，由此将开始一个痛苦的部署过程，可能长达几小时（见图1-1）。由于这个过程太麻烦，所以可能两、三周才会部署一次。

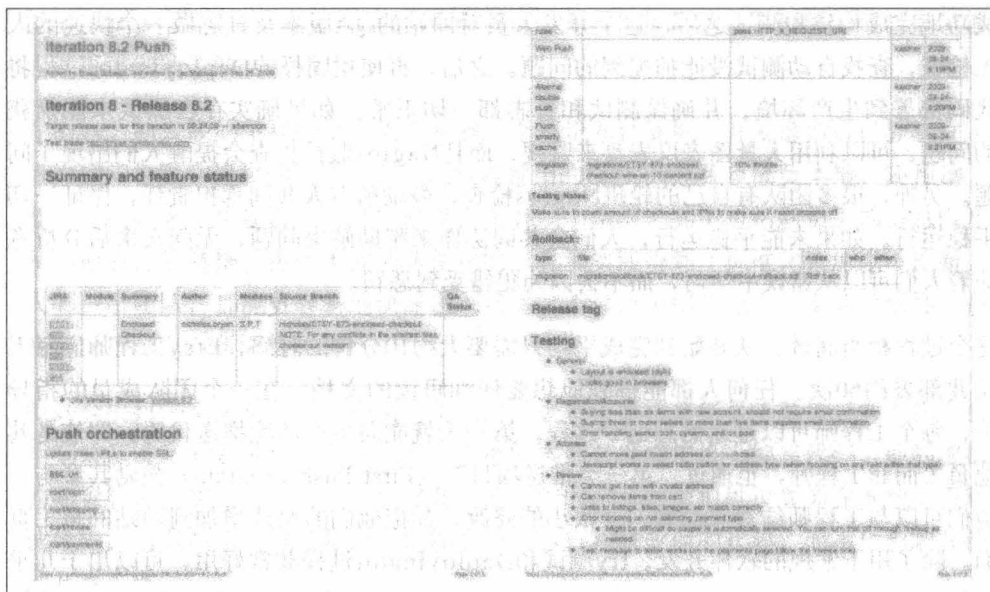


图1-1：有Deployinator之前，部署过程很复杂而且容易出错

逐渐地，人们开始无法忍受这个过程。他们意识到必须要做出一些改变。部署状况实在太糟糕了。这家公司不乏聪明能干的人，他们原本动力十足，如今却深受这个问题的困扰。他们得到了CEO和CTO的一致支持，这对于他们得到资源来做出改变非常关键。

最开始由一个运维工程师向两个开发人员分享部署领域的核心知识，让这些开发人员有时间、有基础根据他们的需要调整这个过程。他们说，一个手里有榔头的人，看到的一切都是钉子，如果有一个Web应用开发人员，那么所有问题看起来都需要一个Web应用，第一个Deployinator就此诞生(见图1-2)。最初，这是一个包装现有shell脚本的Web包装器，不过，经过一段时间后，更多的人加入进来对它进行改进。尽管完成具体工作的底层机制有变化，但总的界面基本保持不变。

所有人都清楚地看到，通过允许这些员工创建工具来改进他们的工作，会让完成类似工作的人轻松得多。原先人们不愿意部署，认为部署很碍事，现在部署则能帮助他们达成目标，更好地为用户展示产品的特性。测试也不再是浪费时间，而是可以帮助捕捉bug。基于日志、图表和提醒，所有人（而不只是少数的几个人）都能看到他们的工作产生的影响。

关于所有这些工具的故事，我们得到的重要收获不是工具本身的特性，而是人们意识到需要构建这样一些工具，而且要提供时间和资源来构建这些工具。



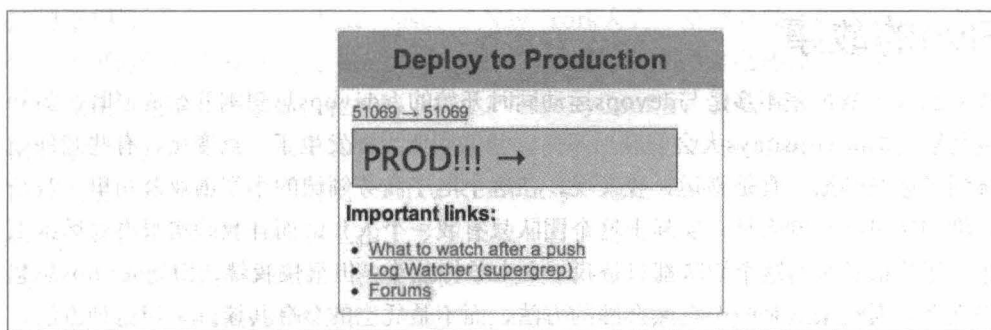


图1-2：有Deployinator之后，任何人都可以使用一个简单Web界面



一方面得到了管理层的支持，可以自由地试验和处理不直接面向客户的代码，另一方面不同团队之间的信任使得这些工具得到有效开发，而且伴随着这些工具的开发，也形成了一个融合工具、分享和协作的文化。

这些要素使Etsy成为了如今人们称道的“devops独角兽”（尽管他们更愿意把自己称作“sparkly horse”），而且维护这种文化也已经成为公司的重中之重。这个例子充分体现了我们对“有效devops”（effective devops）的定义，一个组织利用文化变革来影响个人如何考虑工作、评价个人的不同角色，提升业务价值，并度量改变所产生的影响。正是devops的这些原则帮助Etsy取得了现在的发展，从一个充满挫败感、相互割据的状态变成了业界知名的协作组织和工具构建者。尽管具体细节可能各不相同，但我们已经看到这个行业很多成功故事中都体现了这些指导原则，为了分享这些原则，我们将为希望实现类似转变的组织提供一个指南。

## 这个故事的意义

每个人都按自己的思维方式思考，但概念是共同的。

——Stephen Toulmin, 《Human Understanding》

本书包含来自团队以及个人的案例和故事。查看现在市面上的devops书时，我们发现很少有第一手的实际经验供人们参考学习，很多故事都只关注某一个工具或者一个抽象的文化实践。尽管可以在理论上介绍如何完成一个工作，不过一件事理论上如何做与实践中具体如何操作往往有很大差别。我们希望分享这些实践的真实故事，其中有成功也有失败，我们还会介绍做出这些决策的思维过程，使人们在自己的devops旅程中能得到尽可能多的信息。

## Ryn的故事

我的devops故事差不多是与devops运动同时开始的。devops思想刚开始成形时，另外也是第一届devopsdays大会召开后不久，我的职业生涯发生了一点变化，有些意外地转到了运维领域。真是幸运，我发现，在这个电子商务领域的小型创业公司里，我是运维团队里唯一的女性（实际上这个团队只有我一个人），而且我确实很喜欢运维工作。尽管很长时间这个团队都只是我一个人单打独斗，但很快我就认识到devops思想的意义，看起来这是一个自然合理的方法，而不是凭空的夸夸其谈，采用这种方法，可以与公司里的其他团队更有效地合作。当时，我是一个脾气暴躁的系统管理员，每天都在数据中心忙得晕头转向。我必须随时待命，没有人和我轮班，我每天都忙于四处救火，对于开发人员（或者实际上是任何人）在做什么几乎一无所知。所以分担责任、分享信息以及打破团队间壁垒的想法对我来说实在太合适了。

在接受改变和新想法方面，有些组织相对来说可能更为开明，可是这家创业公司却非常抗拒改变，不仅如此，他们根本不愿意听一个初级系统管理员说什么。“你甚至不是一个真正的系统管理员”，他们以此否决我的想法。另外没有足够的预算，甚至不能给我买几本书（我自己掏钱买了Tom Limoncelli的《The Practice of System and Network Administration》和《Time Management for System Administrators》，这两本书真是物有所值），更不可能派我去参加LISA或Velocity大会，只依靠这家公司我也不可能参加之后两年的devopsdays New York大会。

幸运的是，我开始在网上寻找devops社区，与别人交流并从中学习，与我一样，他们也对运维充满了热情，这些学习和合作激发了我的动力。James Turnbull（现在是Kickstarter的CTO，那时在Puppet工作）在推特上找到我，和我很好地聊了聊，还给我发了他的一本书，就是那本非常棒的《Pro Puppet》，当时我正在与遗留的200台雪花服务器做斗争，甚至没有一个bash脚本来管理这些服务器。这个小小的举动让我认识了一个积极发展、充满活力的社区，使我希望将来有一天我也能成为这个社区的一员。

Jennifer在她的故事里也会提到，如果你整天忙得焦头烂额，将很难真正做出改变。尽管一直在尝试帮助和改进我当时的公司，但由于我缺乏经验（虽然经验在不断增长），我的努力屡屡碰壁，大约一年之后，我终于决定离开。尽管我在不断学习和提高我的技能，但始终感觉不能完全胜任我的工作。看起来我是在与我的同事和组织做斗争，而不是与他们协作。



2013年1月，我参加了第一届devopsdays New York大会，全身心投入地参与了所有讨论，尽可能倾听走廊交流会上大家的发言，不过那时我觉得自己还没有足够的经验，还不能在讨论中发表意见。在推特上关注#VelocityConf后，我从别人那里获得了丰富的经验，并身体力行。那年10月，我在第二届devopsdays New York大会上做了一个闪电演讲，通过这个演讲结识了Mike Rembetsy，他是那一届大会的组织者之一。他告诉我应该去Etsy工作，但是由于多年来我一直认为自己在这个领域就像一个“冒牌货”，我想他只是在开玩笑。在我最初发现运维和devops社区之后，我已经在线关注了Code as Craft和Etsy运维团队，不过我并不认为有足够的能力加入他们的行列。

事实证明我错了，这让我大喜过望。我的运维生涯带我踏上了一个新的旅程，我经历过多种不同的组织结构，体验过开发、运维有时甚至“devops”团队的不同合作方式。我在不同规模的组织里工作过，小到只有25个人的创业公司，大到有数十年历史、拥有几十万员工的企业组织，我还见过开发与交付软件和系统的多种方法，有些比较有效，有些则不然。

近一年的时间里我都是一个人连续待命，还遇到过另外一些不太理想的工作场景，我想把这些年来对我和我的团队最适用的技术和方法分享给大家，有些人可能认为是自己导致了组织的失败，我希望能让人们减少这种想法。写这本书的主要目的是讲述这些故事，这包括我亲身经历的故事，以及其他人的故事，作为一个社区，这样我们就能一同分享、学习和成长。这个社区帮助我有了今天的进步，这本书也是我对社区的一个回馈。

## Jennifer的故事

2007年，Yahoo管理层联系我，问我是否对一个有“一些开发”和“一些运维”的职位感兴趣，这是一个高级服务工程师（Senior Service Engineer）职位，要构建一个多租户、托管、基于地理位置复制的分布式键/值数据库，名为Sherpa。

作为Yahoo的一名服务工程师，我不断磨练自己在编程、运维和项目管理方面的技能。我与构建Sherpa的开发和质量保证团队一起努力，协调数据中心、网络、安全和存储团队的所有工作。2009年，devops之风吹到Yahoo时，我充分体会到这些工作的价值，因为我已经是一个devops了！

“快进”到2011年的夏天，Jeff Park开始接管领导我所在的团队。他帮助这个团队进一步壮大，服务工程组的很多人有些在美国，有些则分布在印度。不仅如此，作为一个老员工，我总在不停歇地工作，而且几乎总是单枪匹马地保证服务正常运转，这让



Jeff很担心。他对业务也很担心，希望通过引入冗余的人员使支持模式有一定弹性。12月，他让我彻底地休假，不要看email，把手机关机，断开与外界的联系。

我告诉他，我觉得有些地方不太对，有些方面可能不能像预期的那样工作。他却告诉我，如果我不好好休息，他就解雇我。他再三向我保证一切都不会有问题。在我度假的前一天晚上，我用JavaScript建立了相关指标的一个简单的可视化显示，另外建立了一个在cron作业上运行的Perl脚本，相信这足以提供警告。

我度假回来后，看到的是一个退化的服务。这些年来我发现的诸多小问题在此期间纷纷出现，对整体产生了极大影响，使服务更难调试。我感觉完全失败了，尽管我最后建立的可视化显示对于找出和监视问题发挥了重要作用。

Jeff把我拉到一边，说他知道在我度假期间很有可能出问题，而且由于团队之前完全依赖我，这也带来了额外的风险。我的“英雄主义”掩盖了系统内在的问题。

他认为，即使短期出现倒退，倘若之后能把这些倒退作为教训，纠正为正确的做法，从长远来看有时这些倒退也是件好事。如果出了问题，这有助于强调知识分享的重要性，需要进一步分享、记录和发布我的业务知识和经验。最终，这会让组织以及团队中的人员更加稳定，总体上得到更好的结果。

这件事让Sherpa团队团结在一起，因为我们要努力修复服务，还要了解发生了什么。我们划分为跨职能的团队来解决问题的不同方面，包括故障处理、沟通组、工具、监控和清理人员。管理层的重要人物一直给予支持，并随时准备做出艰难的决定。这些决定帮助我们大大缩短了总的停机时间。

失败很痛,但能通透你心。

——Bob Sutton, 斯坦福大学管理讲师

这件事带给我的最重要的收获就是失败的意义。我们不能害怕失败，而需要从失败中学习。我们不断组织会议来解决这次事件中突显的运维问题，并继续作为一个交叉学科团队解决问题，这些活动并不仅限于服务工程团队。此外，我们还大力促进消费者与提供者之间的讨论，从而能更好地了解系统的弱点。

十多年来，我一直都是基于运维的部落文化来工作，这包括长时间工作、孤军奋战和尽量避免系统故障，我要如何完成所需的改变呢？

我已经为devops做好了准备。对我来说，devops的意义并不是“开发（devs）做X而运维（ops）做Y，或者开发与运维”之类的说法，而是有关于分享，在整个行业中采

用一种协同的方式解决问题，并加强社区建设。从开放空间到协同处理，一个新的支持系统正在兴起，这将巩固可持续的工作实践以及人与人之间建立的关系。

与Ryn协作完成这本书也增强了我对devops的理解。能够分享世界各地实用的策略和技术，从而帮助改进和建立可持续的工作实践，这实在是一个妙不可言的旅程，而且这个旅程不会因为这本书的结束而结束。

我们每一天都会从很多不同的角度获得大量经验。不论你是刚刚开始你的职业生涯，还是正处于文化转型当中，或者是将要改变你的角色和职责，你的经验都会影响和帮助其他人。希望能听到并与大家分享你的故事，我们可以作为一个社区共同成长，一同从我们的成功和失败中获得经验教训。

## 通过故事描述devops

我们选择了大量案例来描述有效的devops文化会以哪些不同的方式体现。不过，这些故事的目的并不是提供一些模板，要求你严格遵循。如果盲目地复制另一个组织或个人的做法，这会忽视他们做出选择时的具体环境和必要理由。

这些故事只是一些示例或指南。希望你能好好读一读这些故事，并对你的经验进行反思。可能是你现在的经验，也可能是将来得到的经验。我们提供了来自不同方面的很多故事，包括正式的案例研究以及非正式的个人故事。有些故事来自比较知名的组织，不过我们还特意从一些不太有名的组织选择了一些故事，以便向你展示多种不同的devops形式。



阅读这些案例时，不仅要考虑所做的选择和相应的结果，还要考虑它们的具体环境和情境。哪些环境与你自己的环境有哪些类似的地方，另外有哪些关键的差别？如果你在自己的组织中做同样的选择，你的工作环境中的哪些特定因素可能会改变结果？通过阅读和理解这些故事，我们希望你能发现它们深层次的主题，并把它们应用到你自己的devops故事中。

只读这些故事还不够，我们的学习不能就此止步，还要具体实验新的过程、工具、技术和理念。要度量你的进展，最重要的是，要理解相应的原因。一旦认识到哪些尝试适用而哪些不适用，就可以开始做更复杂的试验了。



## 第2章

# 什么是DevOps

devops是一种文化运动，会改变个人对其工作的想法，重视所做工作的多样性，支持加快企业实现价值的过程，以及度量社会和技术改变带来的影响。这是一种思维方式，也是一种工作方式，会促使个人与组织发展和维护可持续的工作实践。这也是一个分享故事和培养同理心的文化框架，允许人们和团队以有效和持续的方式实战运用。

## 文化处方

devops是一个文化处方。任何一个文化运动都不可能存在于真空中；社会结构与文化紧密交织在一起。组织中的层次结构、行业关联以及全球化都会影响文化，还会影响价值观、标准、信念以及反映这些方面的工件。我们创建的软件不能脱离使用和创建这些软件的人独立存在。devops就是要找出方法来调整和变革社会结构、文化和技术，从而能更有效地工作。

## DevOps是多方面的综合

一个标榜自己很新的运动往往有一个危险，它可能试图涵盖所有一切新东西。

——Lee Roy Beach等，Naturalistic Decision Making and Related Research Lines

实现devops并没有一种唯一正确的方法，这本书也不是这样一个“单一处方”。尽管我们会介绍常见的误区和反模式，不过更有兴趣描述成功的devops文化看上去是怎样的，会有怎样的表现，以及这些原则如何应用到各种不同的组织和环境。



尽管devops这个词本身是“development”（开发）和“operations”（运维）合成的一个词，但devops的核心概念可以（而且应当）应用到整个组织。一个可持续的、成功的企业并不只有开发和运维团队。如果加以限制，只考虑具体写软件或者在生产环境部署软件的团队，这会对整个企业带来危害。

## DevOps作为大众模式

在很多方面，devops已经成为一个大众模式，这个词有不同的含义，相应地也存在一些误解。在认知科学领域，大众模式往往作为一个抽象用来表述更具体的想法，而且通常会替代具体想法，与最后讨论的概念相比，这个抽象更容易理解。情境意识就是这样一个例子，这通常用来表示更特定的概念，如感知和短期记忆。大众模式不一定不好，但是不同的组织使用同一个术语却有着不同的内涵时，这就会很成问题。

人们常常花更多的时间争论“devops”是指什么（用来表示怎样的大众模式），而不是花更多时间关注他们真正想要讨论的想法。<sup>注1</sup>有时，为了绕开如何定义devops的问题，人们讨论具体的概念和原则时，会夸大“不好的”行为，由此强调哪些才是“devops”的“好”行为。例如，为了讨论有效的团队间协作，有人可能会用一个漫画例子来说明，展示在一个建立了devops团队的公司中，devops团队只是作为开发和运维团队的中间人（就像我们在前言中所提到的）。这是一个极端的例子，但是这会让人们讨论比定义更有意义、更实用的内容。

## 老观点和新观点

如果有人犯错误就会受到责备和惩罚，在这样一个环境中，会发展一种恐惧文化，这会建起高墙，阻碍清晰、透明的沟通。与之相反的是无问责环境，在这种环境中，问题将通过协作的方式加以解决，而且会把问题看作是个人和组织的学习机会。Sidney Dekker教授在他的《The Field Guide to Understanding Human Error》一书中把这两种环境描述为人为错误的“老观点”和“新观点”。<sup>注2</sup>

第一种环境把“人为错误看作是问题的根源”。这种“老观点”可以描述为一种思维模式，其重点在于消除人为错误。错误是“坏苹果”造成的，应当把这些坏苹果扔掉。这种观点常见于问责文化中，因为它认为错误通常是由于恶意或者能力不足造成的。对失败负责的个人必须受到责罚（或者直接开除）。

注1： Sidney Dekker和Erik Hollnagel, “Human Factors and Folk Models.” Cognition, Technology & Work 6, no. 2 (2004): 79–86。

注2： Sidney Dekker, The Field Guide to Understanding Human Error (Farnham, UK: Ashgate Publishing Ltd, 2014)。

第二种环境将“人为错误看作是一种症状，指示系统可能存在更深层次的问题”。这种“新观点”表述了这样一种思维模式，它把人为错误看作是结构性问题而不是个人问题。人们会根据具体的上下文和对他们来说最有意义的方面做出选择并采取行动，而不是有意的恶意行为或能力不足。组织在考虑尽可能减少问题或对问题做出响应时，应当对系统做全面的考虑。

了解并掌握这种“新观点”是理解devops运动的关键。这种观点鼓励我们分享故事，因为所有情况都是学习机会。

分享故事可以：

- 增进团队中的透明度和信任。
- 指导同事避免成本很高的错误，而不需要他们亲身体验这个错误。
- 增加解决新问题的可用时间，从而可以有更多创新。

通过在全行业中分享这些故事，我们会影响整个行业，创造新的机会、知识和共同理解。

## devops组合

devops的核心是人，人们不仅作为小组，更是作为需要相互理解的团队来工作。这可以描述为一个组合，团队要一起工作，沟通他们的想法以及遇到的问题，并动态地做出调整，朝着共同的组织目标努力。

### 组合示例

通过分析两个攀援者的沟通、澄清和相互信任，可以形象地描述这个重要的组合。攀援活动要求攀援的人在天然岩石或合成墙上向上、向下或左右攀行。共同目标是到达顶端或者到达某个特定路线的终点，通常攀援过程中不能掉落。这是体力和脑力的结合，一方面要有攀爬所需要的耐力，另一方面还要有理解能力及做好下一步准备的敏锐判断力。

在某些攀援活动中，一个人（即攀援者）会用一个绳子和一个背带作为保护措施防止掉落。第二个人（即保护者）会监视绳子的张力，要为攀援者提供足够的张力，一方面防止掉落幅度太大，另一方面还要提供足够的松弛度，使他在攀援时有灵活机动的空间。

要提供适当而安全的保护，不仅要求对工具和过程有共同的理解，还需要持续不断地沟通。攀援者要安全地打好背带，保护者则要确保他的保护设备与攀援者的攀援背带正确连接。开始攀援之前，每个人都要信任对方，不过还要检查另一个人的工作状态。

攀援有一套自己的口头提示，可以在攀援前指示准备情况，攀援者问“on belay?”（保护就绪了吗？），保护者回应“belay on”（保护就绪）。然后攀援者再回应“climbing”（准备攀援）来表示他已经准备好。最后，保护者回应“climb on”（开始攀援）作为确认。

保证这个组合有效工作的原则包括：

- 明确定义的共同目标。
- 持续不断的沟通。
- 对理解的动态调整和修正。

接下来我们会谈到，这些原则不仅对攀援者很有意义，对于具体实现devops的人也具有同样的意义。

## Devops组合示例

Sparkle公司的两个员工分别在不同的团队里工作。General是一个高级开发人员，工作背景很丰富，有很多不同的工作经历，他在Sparkle公司已经任职两年。George是一个运维工程师，有一些经验，在Sparkle公司里还算是一个新员工。

他们所在的两个团队要为依靠Sparkle公司网站开展创新活动的全球用户社区提供支持。他们的共同目标是实现一个新特性，提升最终用户的价值，而且希望不要影响这个网站。

由于在公司里更有经验，General要向George明确Sparkle公司的预期、价值观和目前的流程。反过来，George要向General明确他什么时候需要帮助或者不能理解流程的哪一部分。在继续接下来的步骤之前，General和George要迁入彼此的工作，这正是攀援过程中描述的信任但确认（trust-but-verify）模型的一个例子。

General和George对他们的目标有共同的理解：

- 实现一个新特性，增加对Sparkle公司客户的价值。
- 在相互沟通中保持安全和信任。



在一个非devops的割据环境中，往往缺乏共同的理解，这可能表现为General打算开始编写代码，但未能确保George确实理解需求，也许最后可以实现这个新特性，但是由于对目标没有沟通，前景可能不容乐观。

一个组织在发展过程中肯定会遇到意外的问题或阻碍，不过如果有共同的理解，认识到每个人都是这个组合的一部分，就会采取行动完成一系列的修正。对于谁完成某个特性或者什么时候完成我们可能有一些误解，首先要修正这些误解。另外对于软件表现如何我们有自己的理解，可能存在一些bug会影响我们的理解，因此需要修正这些bug。如果生产环境中的实际工作与我们预期的不同，还要修正流程和相应的文档。

在这本书中，我们将采用这种devops组合的观念，介绍如何利用devops的技术和文化来发展和维护这种共同的相互理解。

# DevOps的历史

通过分析这个行业的历史以及其中反复出现的模式和观念，可以帮助我们理解是什么促成了devops运动。有了这个理解，我们就能了解目前所处的位置，认识到如何通过有效的devops打破循环，避免专业化的不断增加，毕竟这会带来壁垒，还会使某些角色被弱化。

## 开发人员作为运维人员

开始时，开发人员就是运维人员。第二次世界大战爆发时，某国政府号召数学专业人员变成“计算机”，这个工作要求他们负责计算弹道表来评估作战效果。Jean Bartik是响应这个号召的众多女性之一。她的大学导师建议她不要考虑这个号召，希望她延续家族的教育传统，认为这种重复性的工作没有继续深造有意义。

这种数字计算工作确实是重复性的，这一点她的导师没有说错，不过这个工作让Bartik在适当的时间、适当的地点成为了ENIAC (Electronic Numeric Integrator and Computer) 的首批程序员之一，这是第一个全电子可编程计算机系统。

在没有任何文档和任何计划的条件下，Bartik和另外5位加入ENIAC的女性通过检查设备的硬件和逻辑图明确了如何为它编程。要为这个机器和它的18000个真空管编程，这意味着需要通过40个控制面板设置转盘和更换电缆接头。

当时，业界主要关注硬件工程，不太重视保证系统正常工作所需的编程。出现问题时，硬件工程师就会抱怨“不是机器有问题；是操作人员的问题”。程序员深深感到管理和操作这些系统的痛苦，因为他们必须更换熔断器和线缆来消除系统中的bug。

## 软件工程的出现

1961年，约翰·肯尼迪总统提出，希望十年内能有美国人登月，而且还要让他安全地返回地球。面对这个最后期限的同时，具备必要技能的人员却很匮乏，美国国家航空航天局（National Aeronautics and Space Administration, NASA）需要找到合适的人来编写完成这个任务所需的机载飞行软件。NASA任命麻省理工学院（Massachusetts Institute of Technology, MIT）的一位数学家Margaret Hamilton来领导这个项目。<sup>注1</sup>

Hamilton回忆到：

提出新想法就是冒险。奉献和义务是理所当然的。大家都相互尊重。由于软件是一个神秘的黑盒子，管理层给予了我们充分的自由和信任。我们必须找出办法，而且我们确实做到了。回首过去，我们真是世界上最幸运的人；那时别无选择，必须做开拓者；没有时间容我们从头开始学习。<sup>注2</sup>

在努力编写这个复杂的软件期间，Hamilton还有一大创举，她发明了软件工程这个词，另外她还建立了优先级显示的概念，这个软件会提醒宇航员需要他们实时关注的信息。她提出了一组需求，在软件工程问题列表中加入了额外的质量保证，这包括：

- 调试所有单个组件。
- 在组装之前测试单个组件。
- 完成集成测试。

1969年，在阿波罗11号任务中，月球模块导航计算机软件计算任务过多，已经超出它有限的能力范围。Hamilton的团队调整了这个软件，将它写为可以人工覆盖，允许尼尔·阿姆斯特朗介入，可以利用人工控制操控月球模块。

一方面管理团队对编写机载飞行软件的工程师团队赋予了充分的自由和信任，另一方面团队成员之间也互相尊重，由此得到的软件最终使尼尔·阿姆斯特朗能够在月球上迈出了一小步并且让人类在技术上迈出了一大步。如果没有这种高度信任的环境，这种人工覆盖功能（最后证实这是至关重要的）可能根本不会有，而登月故事可能会有完全不同的结果。

---

注1： Robert McMillan, "Her Code Got Humans on the Moon—And Invented Software Itself," WIRED, October 13, 2016.

注2： A. S. J. Rayl, "NASA Engineers and Scientists—Transforming Dreams Into Reality," 2008, [http://www.nasa.gov/50th/50th\\_magazine/scientists.html](http://www.nasa.gov/50th/50th_magazine/scientists.html).



## 软件的问题

不只是在太空飞行领域，实际上，在20世纪60年代，各个领域软件都开始变得越来越重要。随着硬件越来越容易得到，由于软件没有遵循其他工程学科中的标准，人们开始越来越担心软件的复杂性。系统的增长速度以及出现的这种依赖性很令人担忧。

1967年，北约科学委员会（由世界各国不同行业的科学家组成）展开讨论，评价软件工程的现状。1967年秋天成立了一个计算机科学方向的研究组，其目标就是重点关注软件的问题。他们邀请了各个行业领域的50位专家，形成3个工作组，分别关注软件设计、软件生产和软件服务，着力定义、描述并开始解决软件工程的问题。

在1968年的NATO软件工程大会上，明确了软件工程的关键问题，包括：

- 定义和度量成功。
- 构建需要大量投资而且可行性未知的复杂系统。
- 按进度和规范生成系统。
- 对构建特定产品的制造商施加经济压力。

这些问题的明确对定义和形成之后数年这个行业的重点领域很有帮助，而且直到今天对我们仍有影响。

## 专有软件和标准化的出现

直到1964年，通常的做法是制造专门针对客户需求的特定计算机。软件和硬件不是标准化的，不可替换。1964年，国际商用机器公司（International Business Machines, IBM）宣布推出了System/360计算机系列，这些计算机设计用来支持规模从小到大的各种功能，可以用于商业以及科学领域。

其目标是减少产品开发、制造、服务和支持的成本，同时允许客户根据需要升级。System/360成为主导的大型计算机，为客户提供了相当大的灵活性，可以先从小规模开始，根据需要扩展计算资源。它还使工作有了灵活性，个人可以学习软件和硬件（在当时软硬件是紧耦合的），从而拥有必要的技能，在其他地方也能完成类似的工作。

在20世纪60年代末之前，计算机都还是租用的，而不是直接购买。硬件的成本相当高，另外还要考虑软件和服务的成本。软件的源代码是公开提供的。1969年，由于一项美国反垄断诉讼，IBM将其产品的软件和硬件解耦合，对与其大型机硬件关联的软件单独收费，再一次对这个行业带来了巨大影响。这改变了人们对软件的看法，软件本身突然之间获得了显著的货币价值，因而不公开提供。

## 网络时代

1979年，Tom Truscott和Jim Ellis启动了一个全世界范围的分布式讨论平台，名为Usenet，那时他们还是杜克大学的学生。开始时Usenet只是一个简单的shell脚本，可以自动呼叫不同的计算机，搜索这些计算机上文件的变化，然后使用UUCP (UNIX-to-UNIX拷贝程序，这是一组支持计算机间传输文件和执行远程命令的程序)将这些变化从一台计算机复制到另一台计算机。Ellis在一个名为USENIX的UNIX用户组发表了题为“Invitation to a General Access UNIX Network”<sup>注3</sup>的演讲。这是在组织之间利用计算机沟通和分享知识的最早的方法之一，这种方法迅速得到普及。

这个工具促进了大学和公司之间的知识分享，而与此同时，在这个阶段，关于公司如何运营的细节通常都属于他们的“商业秘密”。不会在公司以外讨论如何解决问题，因为这些知识被视为竞争优势。当时的公司文化往往会有意让竞争对手低效地工作。这大大妨碍了协作，严重限制了可用沟通渠道的有效性。这种文化壁垒导致公司的复杂性大大增加。

越来越复杂的系统进一步要求技能和角色的特殊化、专业化。这些角色包括系统管理员（专门负责系统管理和尽可能减少系统开销）和软件工程师（专门负责创建新产品和新特性来解决新的需求）。另外一些更专业的小组同样彼此割据，网络运营中心 (network operations center, NOC), 质量保证 (QA)、安全、数据库和存储都变成孤立的问题领域。

这种情况就导致组织里建起一座“巴别塔”，大家都自行其是，由于关注不同的问题，所以各自都在讲不同的语言。除了存在这种壁垒，软件与运行这些软件的硬件分别有不同的特定问题，二者是分开的。开发人员不再为修复崩溃的系统熬到深夜，也不再直接面对不满意的用户，承受他们的抱怨。另外，编程朝着更高级语言的趋势发展，这意味着开发变得更加抽象，与硬件和过去的系统工程师离得越来越远。

---

注3： Ronda Hauben和Michael Hauben, Netizens: On the History and Impact of Usenet and the internet (Los Alamitos,CA: IEEE, 1997)。



为了预防和避免服务出现问题，系统管理员要用文档记录人工完成常规操作所需要的步骤。系统管理员从全面质量管理（total quality management, TQM）借用了“根本原因分析”（root cause analysis）的思想。在某种程度上这使人们开始关注并努力减少风险。由于缺乏透明度和变更管理，这带来大量问题，工程师必须处理越来越多这样的问题。

## 全球社区的起源

互联网使程序员和IT从业人员可以在线分享他们的想法，不仅如此，人们开始想方设法面对面交流他们的想法。用户组就提供了这样一个途径，在这里不同技术的从业人员和用户可以当面讨论他们的领域，用户组的数量和加入的人数都开始增长。其中全世界最大的用户组之一是数字设备计算机用户协会（Digital Equipment Computer Users' Society, DECUS），这个用户组于1961年成立，成员主要由为DEC计算机设备编写代码或维护这些设备的程序员组成。

DECUS美国分部举办了大量技术会议，并在美国成立了很多本地用户组（local user groups, LUGs），其他国家分部也在世界各地做着同样的努力。这些会议和活动开始以DECUS会议论文集的形式发表他们的文章和想法，用户组成员可以得到这些论文集，这也成为分享信息和发展社区知识以及建立成员联系的一个途径。

USENIX与它的特别兴趣组建立了一个专门面向系统管理员的类似社区，即系统管理员兴趣组（System Administrators Group）。后来这被称为SAGE，如今这个兴趣组被称为大规模服务器环境中的系统管理（Large Installation System Administration, LISA）特别兴趣组，每年都举办同名的大会<sup>注4</sup>。另外，NSFNET“地区技术”会议演变成成为北美网络运营商联盟（North American Network Operators' Group, NANOG），这个社区特别面向网络管理员，通过增加协作使互联网变得更好。

这些本地和全球性用户组的一个主要特点是关注知识分享，与之相反，当时的技术公司则往往把实践做法视为秘密。为了追求自己的财务和物质成功，公司会把他们的过程作为严守的秘密，因为如果竞争对手的做法不那么高效，这些公司取得成功的可能性就会更大。公司非常不提倡员工在行业大会上分享知识，甚至明确禁止这样做，试图以此保持这种竞争优势。这与更近的发展形成鲜明反差，如今社区和大会都在不断发展知识分享以及公司之间的交叉协作。

---

注4：USENIX已经宣布LISA将于2016年底解散。有关的详细信息参见<https://www.usenix.org/blog/refocusing-lisa-community>。





## 行业秘密和专有信息

如果信息有足够的私密性，可以带来经济或商业优势而不能公开，这就被认为是一个行业秘密。而公司拥有、所有或持有专有权的信息则认为是专有信息。软件、过程、方法、薪酬结构、组织结构和客户名单都是公司专有信息的例子。例如，专有软件是指不向最终用户提供源代码的软件。所有行业秘密都是专有的，但并不是所有专有信息都是秘密。

除了行业文化的变化，商业化以及知识和技术的成本也会影响哪些公司会在其组织中保守秘密。

## 应用和Web时代

作为早期跨组织边界成功协作的一个例子，Apache HTTP服务器于1995年发布，这个Web服务器非常流行。基于Robert McCool开发的公共域NCSA HTTP守护程序（当时他还是伊利诺伊大学厄巴纳-香槟分校的一个本科生），这个模块化Apache软件允许任何人利用最少的配置快速部署Web服务器。这是一个开端，标志着向这种开源解决方案发展已成趋势。开源软件（即已经提供许可，允许用户读、修改和发布其源代码）开始与封闭的专有源码解决方案展开竞争。

由于可以得到Linux操作系统的各种版本，加上诸如PHP和Perl等脚本语言越来越普及，开源运动带来了LAMP栈（通常一起使用的Linux、Apache、MySQL和PHP）的蓬勃发展，这是构建Web应用的一个非常流行的解决方案。MySQL是1995年首次发布的一个关系数据库，结合PHP的服务器端脚本功能，允许开发人员创建动态网站和应用，与以往相比，这些动态网站包含可以更快地更新或者能够动态生成的内容。由于创建这些新的Web应用相当容易，20世纪90年代后期，人们以及组织必须更高效、更灵活地工作，以保持其竞争力。

在这个时期，系统管理员和计算机程序员都倍感焦虑和困惑。在系统管理领域，长期以来一直有这样一种文化，就是总是说“不”，强调“保持稳定性至关重要”。1992年，Simon Travaglia开始在Usenet上发表一系列文章，名为The Bastard Operator From Hell (BOFH)，其中描述了一个糟糕的系统管理员会把他的困惑和愤怒发泄到系统的用户身上。这种“有毒”的操作环境会让该领域的一些人把这个糟糕的系统管理员看作是英雄，并模仿他的行为，通常这又会危害到他们周围的其他人。

在开发领域，则有这样一种文化：“这些改变至关重要”和“我不想知道这该怎么做，因为过于深究会让我停滞不前”。有些环境中，这种草率行事的开发人员会危害

到系统的稳定性，他们会寻找一些非官方的旁门左道绕过既定的流程来达到他们自己的目标。这会进一步带来更大范围的调整，使得人们越发认为改变很有风险。在团队中，如果有人想要对整个流程做出改变，通常会发现自己陷入了一个泥潭，他将变成一个主题专家，因为某些维护工作变得至关重要，他必须固守在这些位置上提供支持而无法脱身。

## 软件开发方法论的发展

2001年，对极限编程（Extreme Programming，XP）感兴趣以及这个社区中活跃的一些人收到了一份邀请，请他们共同讨论软件开发，收到邀请的还包括这个领域的其他人士。极限编程是敏捷开发的一种形式，与之前的软件开发方法论相比，对于不断改变的需求，极限编程更有响应性，并以其较短的发布周期、广度测试和结对编程而闻名。17位软件工程师接受了这个邀请，他们聚集到美国犹他州的滑雪胜地Snowbird滑雪场，总结了他们共同的价值观，指出希望在开发中看到对变化的适应性和响应性，并且特别把重点放在人为因素上。这份敏捷宣言（Agile Manifesto）发起战斗号角，宣告了敏捷运动的开始。

2004年，软件开发人员Alistair Cockburn（他也是敏捷宣言的合作者之一）描述了水晶项目开发方法（Crystal Clear）<sup>注5</sup>，他对成功团队做了10年的研究，并基于这些研究提出了这种针对小型团队的软件开发方法论，其中描述了3个共同特征：

- 经常交付可用代码，趋向于更小规模、更频繁的部署，而不是不经常的大规模部署。
- 反思改进，或者对之前工作中好的和不好的方面进行反思，帮助指导将来的工作。
- 开发人员之间的渗透式交流，其想法是如果开发人员在同一个房间里，信息将流向团队成员的背景听觉，使成员就像通过渗透一样获取相关信息。

这个运动在软件开发领域持续多年，之后影响进一步扩大。当时一个名叫Marcel Wegermann的系统管理员写了一篇文章，讨论如何利用Crystal Clear、Scrum和Agile的原则，并将这些原则应用到系统管理领域。他对这个主题做了一个闪电演讲，其中提出了一些建议的想法，如对Linux操作系统的/etc目录进行版本控制、实现系统管理

---

注5： Alistair Cockburn, Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams (Boston: Addison Wesley, 2004)。



结对以及操作反思，除此以外，他还于2008年建立了敏捷系统管理邮件列表（Agile System Administration mailing list）。

## 开源软件，专有服务

随着开源软件的蓬勃发展以及软件整体变得越来越模块化和可互操作，工程师发现他们在工作中有了越来越多的选择。开发人员不再局限于某个硬件开发商或某个硬件支持的操作系统和专有软件，现在开发人员可以选择他们想使用的任何工具和技术。随着软件（尤其是Web软件）变得越发商业化，软件的价值同时变得既高又低，一方面软件不再那么专有，变得更通用，但同时软件开发人员薪酬很高，而且对软件开发人员的需求量很大。

2006年，Amazon.com, Inc（一家电子商务公司，那时通常认为它是一家向顾客出售图书和其他商品的网站）上线了两个服务：亚马逊弹性计算云（Amazon Elastic Compute Cloud, EC2）和亚马逊简单存储服务（Amazon Simple Storage Service, S3），这是首个开创性的尝试，着力通过专有服务提供虚拟化计算实例和存储。这使得个人可以快速地运用计算资源，而不需要前期投入大笔的硬件开支，而且可以根据需要请求更多计算资源。就像当初引入System/360一样，这个服务很快得到采用，由于其易用性、低准入成本和灵活性，它逐渐成为事实上的标准。

随着Web技术继续发展和演变，人们在线交流和协作的方式也在发展。推特（Twitter）作为一个在线社交网络服务于2006年问世。最初看起来这个工具主要面向那些希望用短格式分享信息和专注时间短的人，或者帮助名人与他们的粉丝互动。不过，到2007年，推特的使用量呈爆炸式增长，这要归功于西南偏南互动大会（South by Southwest Interactive, SXSW）在走廊的大屏幕上实况发推特转播大会情况。

推特很快成为在全球范围内建立特别社区的一种方法。对于会议，利用推特可以得到多声道系统无法提供的价值，并且可以让志趣相投的人建立联系。“走廊交流会”通常用来描述人们在会议走廊上的互动和交谈，如今已经从真实世界扩展到Web网络，任何人都能发现并参与这些特别的互动。

## 敏捷基础设施

在大多伦多的Agile 2008大会上，系统管理员和IT咨询师Patrick Debois在他的演讲“Agile Operations and Infrastructure: How Infra-gile are You?”中谈到可以将Scrum结合到运维中。Patrick在一个测试数据中心迁移的项目中与开发和运维团队一起工作。



在他的工作中，需要频繁往返于开发团队和运维团队之间，可能头一天在进行开发，第二天又要与运维团队一起忙于救火，这种来回切换让他疲于奔命。实际上，如果不能只专注于一个任务，即使只是在两个任务间来回切换，这种任务切换的开销也会导致生产力下降近20%<sup>注6</sup>。

同样在这个大会上，Andrew Clay Shafer（之前他是一个软件开发人员，后来开始对IT问题产生浓厚兴趣）提出设立一个敏捷基础设施分组会议。不过，他认为没有人会对这个主题感兴趣，所以最后并没有参加他自己提出的这个分组会议。Patrick看到后，意识到并非只有他一个人对敏捷系统管理感兴趣，并在会外联系了Andrew，对这个概念进行了更深入的讨论。

大约在同一时间，多个公司也开始关注这个领域，不仅在流程方面取得长足进步，从而能够跟上互联网越来越快的变化，同时还通过围绕一些很受欢迎的会议（如O'Reilly Velocity大会）成立的社区公开分享他们的一些故事。

Flickr就是这样一家公司，这是颇受摄影爱好者好评的一个社区网站。2005年被Yahoo收购后，Flickr需要将它的所有服务和数据从加拿大迁移到美国。John Allspaw对Web运维有着无比的热情，在系统运维领域工作了多年。他作为运维工程经理加入了Flickr公司，帮助完成这个新的迁移项目。Paul Hammond在2007年加入Flickr开发团队，2008年成为Flickr工程经理，与Allspaw合作领导这个开发组织。

在美国圣克拉拉举办的Velocity 2009大会上，Hammond和Allspaw共同发表了“10+ Deploys per Day: Dev and Ops Cooperation at Flickr”（每天部署10次以上：Flickr的开发和运维合作），强调了促使团队快速前进的革命性变化。他们并不是通过着手打破割据或者发起一个大型专业性文化运动来做到这一点。在Flickr的工作中，他们能够充分地合作，这与Allspaw以往在Friendster的工作经历形成鲜明对比（在Friendster，尽管热情很高，压力很大，但是团队间的合作微乎其微）。



不能因为“每天完成了10次部署”就宣称你“成功地实现了devops”。要注意你的组织所要解决的特定问题，而不能只依赖从其他组织听到的度量指标。应当记住为什么要做这些特定的改变，而不能只看部署次数或者其他的任意指标。

两位经理充分利用了在一起工作的机会。他们两人都不会贸然行事，不会在某一天醒

注6: Gerald Weinberg, Quality Software Management: Systems Thinking (New York: Dorset House Publishing Company, 1997)。

来时突然决定公司需要做一个很大的改变。他们发现，协作完成小部分工作会得到更好的结果。他们把这些小的工作记录下来（这些小的变化最后会累积成为更大的文化改变），这种合作所带来的影响远远超过了部署次数的影响。

## devopsdays的起源

不要只是说“不”，这是对别人问题的不尊重…… #velocityconf#devops  
#workingtogether

——Andrew Clay Shafer (@littleidea)

这条推文是Andrew Clay Shafer在2009年6月23日发的，这让Patrick Debois不禁在推特上遗憾地抱怨：尽管能远程观看，但不能在现场亲身参加那一年的Velocity大会。当时Pris Nasrat是Guardian的首席系统集成师，他回了一条推文，“为什么不在比利时举办一个你自己的Velocity大会呢？”受此启发，Patrick确实这么做了，他开始着手举办一个本地大会，使开发人员、系统管理员、软件工具程序员以及在这些领域工作的其他人能够聚在一起。那一年10月，第一届devopsdays大会在Ghent召开。两周后，Debois写到：

坦率地讲，过去几年中我参加一些敏捷会议时，感觉就像在沙漠里布道。那时我真的有些灰心，一度认为让开发人员和运维人员一起合作的想法可能太疯狂了。不过，现在看来，哇哦！这个概念真有些烈火燎原的感觉。

第一届devopsdays大会就像引燃了火药桶，大家纷纷指出哪些需要未得到满足。人们彼此割据，对devops的现状很失望，认为这种方法没有多少新意，描述的只是他们认为之前已经在做的工作。随着很多人开始在世界各地举办新的本地devopsdays会议，这个大会不断发展和壮大。另外利用推特提供的实时通信，走廊交流会根本停不下来，于是就诞生了#devops。

## devops的现状

自Patrick Debois在比利时举办第一届devopsdays大会以来，可以看到这6年来devops运动取得了显著的进展，这让人很受鼓舞。Puppet发布的2015 devops现状报告中发现，实现devops的公司的表现远远好于不采用devops的公司，并且最后通过数字展示了很多人已有的一个设想。如果把重点放在让团队以及个人高效地合作，与割据的团队相比（而且这些团队里很多工程师无法与别人很好地合作），前者对企业更有利。高绩效的devops组织可以更经常地部署代码，失败更少，而且能够更快地从失败中恢复，另外员工也更有幸福感。

devopsdays大会已经从2009年的1次增加到2015年全世界范围的22次。每年都会在新的地点召开新的devopsdays大会，这已经不是一个仅限于硅谷或纽约之类的技术中心才有的现象。还有数十个本地聚会小组，有数以千计的成员，这些成员遍及全球各个地方，此外推特上每天都有关于这个主题的对话。

## 小结

反思我们的历史，可以看到这样一种趋势：就是更多地强调结果，而没有重视人和过程。很多人从John Allspaw和Paul Hammond的“每天部署10次以上”演讲得出结论，认为重要的是部署数量，一天要部署10次以上，而副标题“Flickr的开发和运维合作”总是被忽略。

如果局限于一个特定的结果，因为组织限制本身而很有压力的人更会倍有负担。与机械加工不同，软件的结果很大程度上依赖于人为因素。有可能软件还没有完成就已经过时，也可能不能满足客户的期望，另外还有可能以意外的方式失败并带来灾难性的影响。

如果把重点放在文化和过程上，这会鼓励迭代和改进完成工作的做法和理由（如何做和为什么做）。把关注点从“是什么”转向“为什么”时，我们就能获得自由和信任，相信我们的工作是有用而且有意义的，这也是工作满意度的一个关键要素。对工作的投入会影响结果，而不是只关注得到某个特定的结果，这样人们会更愉快、更有效率，从而完成人类的下一步飞跃。

devop的引入改变了我们的行业，它把重点放在不同角色的人和过程上，鼓励协同与合作，而与专业化并不抵触。



## 第4章

# 基本术语与概念

关于有效实现devops，要建立一个坚实的基础，首先需要讨论一些关键术语和概念。有些概念读者可能已经很熟悉，还有一些在前面介绍软件工程历史时已经提到，或者如果读者对各种软件开发方法论有一些经验，可能已经对另外一些概念有所了解。

在计算机工程的整个历史中，已经提出了很多方法论来改进和简化软件开发和运维的过程。每个方法论都把工作划分为多个阶段，分别有一组不同的活动。很多方法论存在的一个问题是：它们只关注开发过程，认为这个过程与运维工作是分离的，这就导致不同团队的目标存在冲突。另外，如果强制其他团队采用某些特定的方法论，倘若这个工作与他们的过程和目标不符，就会带来不满和挫败感。应当了解不同的方法论是如何工作的，以及分别会带来哪些好处，这将有助于增进理解并减少这种摩擦。



devops并没有严格定义为禁止使用某个特定的方法论。尽管devops由提倡敏捷系统管理和鼓励开发与运维团队合作的人提出，但实践的细节特定于具体的环境。在这本书中，我们会反复强调一点：devops的一个重要方面就是能够评价和评估不同的工具和过程，从而找出针对你的具体环境最有效的工具和过程。

## 软件开发方法论

划分开发工作的过程（通常划分为不同的阶段）就被称为软件开发方法论。

这些不同的工作阶段可能包括：

- 可交付产品或工件的规范。

- 根据规范完成代码的开发和验证。
- 将代码部署到最终客户或生产环境。

要想在这一章中涵盖所有方法论是不可能的，不过我们会简单介绍对devops基本思想存在某种影响的那些方法论。

## 瀑布方法论

瀑布方法论或模型是一个项目管理过程，其重点是由这个过程的一个阶段到下一个阶段的串行进程。瀑布方法最早发源于制造和建筑行业，后来被硬件工程所采用，20世纪80年代初针对软件对瀑布模型做了相应调整<sup>注1</sup>。

原来的阶段分别是需求规范、设计、实现、集成、测试、安装和维护，这个发展进程可以描述为从一个阶段流向另一个阶段（瀑布模型也因此得名），如图4-1所示。

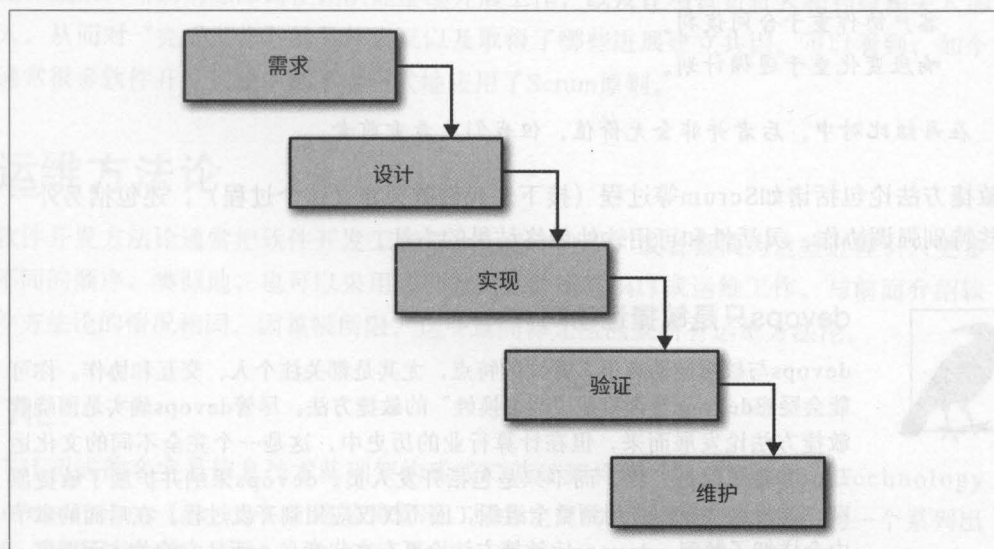


图4-1：瀑布模型

采用瀑布模型的软件开发往往是高度结构化的，大量时间花费在需求和设计阶段，其思想是如果正确地完成这两个阶段，就能大大减少后期可能发现的错误。

在瀑布模型的全盛时期，利用光盘或软盘交付软件成本很高，这还不包括客户人工安

注1： Herbert D. Benington, Production of Large Computer Programs. IEEE Annals of the History of Computing, October 1, 1983, <http://bit.ly/benington-production>.

装的相应成本。如果要修正一个bug，这需要制作和发布新的软盘或光盘。由于存在这些成本，很有必要在前期花更多的时间和精力建立需求规范，而不是到后期再尝试改正错误。

## 敏捷方法论

敏捷方法论实际上是指一组软件开发方法论，与之前的方法（如瀑布方法）相比，这些方法论设计得更轻量级也更加灵活。2001年编写的敏捷宣言（上一章中介绍过）将敏捷方法的主要原则概括如下：

我们通过身体力行和帮助他人来揭示更好的软件开发方式。经由这项工作，我们形成了如下价值观：

个体与交互重于过程和工具。

可用的软件重于完备的文档。

客户协作重于合同谈判。

响应变化重于遵循计划。

在每组比对中，后者并非全无价值，但我们更看重前者。

敏捷方法论包括诸如Scrum等过程（接下来我们就会定义这个过程），还包括另外一些特别强调协作、灵活性和可用软件最终结果的方法。



### devops只是敏捷运动吗？

devops与敏捷运动有很多共同的特点，尤其是都关注个人、交互和协作。你可能会疑惑devops是否只是“改名换姓”的敏捷方法。尽管devops确实是围绕着敏捷方法论发展而来，但在计算行业的历史中，这是一个完全不同的文化运动，其覆盖面更广泛，而不只是包括开发人员。devops采纳并扩展了敏捷原则，并将这些原则应用到整个组织，而不仅仅应用到开发过程。在后面的章节中会详细了解，devops比敏捷方法论更有文化意义，而且它的关注面更宽，并不仅限于交付速度。

## Scrum

20世纪90年代中期，Ken Schwaber和Jeff Sutherland博士（敏捷宣言的其中两位原创者）结合各自的成果提出了一个新的软件开发过程，称为Scrum。Scrum软件开发方法论的重点是最大化开发团队的能力，从而对项目 and 客户需求中的变更快速做出响应。Scrum使用预定义的开发周期（称为sprint），通常在一周到一个月之间，开始时



会有一个sprint计划会议定义目标，最后会有一个sprint评审会议和sprint回顾会议讨论进展以及这个sprint中出现的问题，以此结束这个sprint。

Scrum的一个关键特点是每日站会（Scrum每日例会），这是一个每天都要开的例会，团队成员要分别（相当快地）回答3个问题：

- 昨天我做了哪些工作帮助团队实现sprint目标？
- 今天我计划做什么来帮助团队实现这些目标？
- 我发现哪些问题可能妨碍我或团队达到目标（如果有）？

这些会议在早晨召开，使人们能够对当天计划要做的工作达成一致，并互相帮助解决其他成员可能存在的棘手问题，这些会议通常由流程管理员（Scrum master）召集。流程管理员是一个很重要的角色，他还有另外一些职责，如帮助团队自我组织和协调工作成果，帮助清除障碍使团队能继续开展工作，以及让项目负责人和利益相关人加入，从而对“完成工作”的具体含义以及取得了哪些进展建立共识。可以看到，如今通常很多软件开发实践中都不太正式地应用了Scrum原则。

## 运维方法论

软件开发方法论通常把软件开发工作划分为多个阶段，或者尝试为这些过程引入更多不同的顺序，类似地，也可以采用这种方式划分或组织IT或运维工作。与前面介绍软件方法论的情况相同，因篇幅所限，这一章同样无法涵盖所有运维方法论。

### ITIL

ITIL正式的名字是信息技术基础架构库或IT基础架构库（Information Technology Infrastructure Library），这是为管理IT服务定义的一组实践。ITIL作为一个系列出版，其中包括5卷，描述了其流程、过程、任务和检查表，可以用来说明依从性，并度量朝着这个目标做出的改进。20世纪80年代越来越多的IT组织开始使用越发多样的实践方法，ITIL正是由这种趋势发展而来。

英国国家计算机和电信局（Central Computer and Telecommunications Agency，CCTA）开发了一套建议，着力标准化这些实践方法。ITIL最早于1989年出版，多年来，这些书和实践方法在不断发展，在最新版本（2011版）中，5大核心部分分别描述了服务战略、服务设计、服务转换、服务运营和服务持续改进。

IT分析师和咨询师Stephen Mann指出，尽管ITIL的标准化会带来很多好处，而且全世界已经有超过150万人获得了ITIL认证，但从业人员可能还希望关注另外一些领域。Mann指出，ITIL通常更侧重于被动管理而不是主动管理，所以我们建议使用ITIL的组织能够记录可以尝试哪些方法在他们的实践中增加更多主动规划，并更加关注客户。

## COBIT

信息及相关技术控制目标（Control Objectives for Information and Related Technology, COBIT）是一个用于信息与技术治理和管理的ISACA框架，最早于1996年发布。COBIT的一个核心原则是保证企业目标与IT目标一致。

COBIT的5个主要原则包括：

- 满足利益相关人的需要。
- 全过程覆盖整个企业。
- 应用一个集成框架。
- 支持一个整体方法。
- 区分治理和管理。

## 系统方法论

有些方法论将系统作为一个整体来考虑，而不是只关注更特定的领域，如软件开发或IT运维。系统思考能力对于处理复杂系统的人来说至关重要，如当前创建的很多软件产品都是复杂系统；如果读者有兴趣，希望对一般意义的系统思考有更多了解，可以参考Donella Meadows的《Thinking in Systems》和Dr. Richard Cook的《How Complex Systems Fail》。

## 精益方法

通过对汽车生产和丰田生产系统（Toyota Production System, TPS）5年的研究，James P. Womack、Daniel T. Jones和Daniel Roos创造了精益生产（Lean Production）的概念，<sup>注2</sup>Womack和Jones定义了精益思想的以下5大原则：<sup>注3</sup>

---

注2： James P. Womack, Daniel T. Jones, and Daniel Roos, The Machine That Changed the World (New York: Rawson Associates, 1990)。

注3： James P. Womack and Daniel T. Jones, Lean Thinking (New York: Simon & Schuster, 1996)。

- 价值 (Value)。
- 价值流 (Value stream)。
- 流动 (Flow)。
- 拉动 (Pull)。
- 尽善尽美 (Perfection)。

基于这些想法，尤其是通过系统识别和消除浪费来追求尽善尽美，可以将精益 (Lean) 定义为客户价值最大化和浪费最小化。

精益系统强调系统中通过消除其他地方的浪费而增加价值的部分，这些浪费可能是某些部件的过量生产、必须改造的不合格产品，或者是因等待系统其他部分所花费的时间。由此可以进一步得到精益 IT 和精益软件开发的观念，就是将同样的这些概念应用到软件工程和 IT 运维中。

这些领域中可以消除的浪费包括：

- 不必要的软件特性。
- 通信延迟。
- 过长的应用响应时间。
- 过于繁琐的过程。

精益环境中的浪费与价值正好相对。Mary Poppendieck 和 Thomas Poppendieck 将精益制造的浪费如下映射到软件开发的浪费：<sup>注4</sup>

- 部分完成的工作。
- 额外的功能。
- 再学习/返工。
- 不必要的交接。
- 任务切换。

注4： Mary Poppendieck and Thomas David Poppendieck. Implementing Lean Software Development (Upper Saddle River, NJ: Addison-Wesley, 2007)。



- 延期。
- 缺陷。

与devops类似，完成精益软件开发并非只有一种方法。实现精益有两种主要方法：一种方法强调通过一个工具集消除浪费，另一种方法则关注改进工作流，这也称为丰田方法（The Toyota Way）。<sup>注5</sup>这两种方法有同样的目标，不过由于做法不同可能会得到不同的结果。

## 开发、发布和部署概念

关于软件的开发、发布和部署有很多术语，这一章前面讨论的方法论定义中提到了一些术语，但还有很多术语尚未涵盖。这些概念描述了开发和部署软件的方式方法，很有必要理解这些概念是什么以及它们的相互关系，这会让读者对于如何使用工具全面支持这些实践方法有更充分的理解。

### 版本控制

版本控制系统会记录系统中存储的文件或文件集的变更。这些文件可能是源代码、资产以及软件开发项目中的其他文档。开发人员可以按组完成变更，这称为commits（提交）或revisions（修订版）。每个修订版以及相关的元数据（如谁在什么时间完成了这个变更）会以某种方式存储在系统中。

由于能够在仓库中提交、比较和合并版本，还可以恢复对象过去的修订版本，这使得团队内部以及团队之间可以有更充分的合作和协作。通过建立这样一种方法，能够将生产环境中的对象回退到之前的版本，从而尽可能地减少风险。

### 测试驱动开发

在测试驱动开发中，代码开发人员首先为新代码功能编写一个测试（这个测试会失败），然后编写这个代码本身，最后确保代码编写完成时能够通过这个测试。测试是清晰地定义新功能的一种方法，可以更明确地指出代码应当做什么。

通过让开发人员自己编写这些测试，不仅可以大大缩短反馈周期，还会鼓励开发人员对所编写代码的质量更为负责。这种职责分担以及缩短开发周期一直是devops文化中重要的部分。

---

注5： Jeffrey K. Liker, The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer (New York: McGraw-Hill, 2004)。

## 应用部署

应用部署是计划、维护和执行一个软件版本交付的过程。一般来讲，应用部署手段需要考虑系统底层发生的变化。可以由基础设施自动化构建运行特定应用所需的依赖库（可能是计算、操作系统或其他依赖库），这样可以尽可能减少不一致性对所发布软件的影响。

取决于应用类型，不同的工程问题可能会很重要。例如，数据库在一致性方面可能有严格的要求。如果一个事务发生，就必须在数据中有所反映。应用部署是工程质量软件的一个至关重要的方面。

## 持续集成

持续集成（Continuous integration, CI）是一天中频繁地将开发人员编写的新代码与一个主线或“主”分支集成的过程。这与另一种做法截然不同，即开发人员分别完成独立的特性分支，可能耗时数周或数月，只有当这个独立分支完全完成时才将其代码合并到主分支。由于合并的时间间隔很长，这意味着在此期间可能已经发生更多变化，其中的一些变化造成破坏的可能性会更大。由于变更集更大，隔离和识别导致破坏的原因也会更加困难。通过频繁地合并小的变更集，就能更容易地找出导致回归的特定变更。持续集成的目标是避免不经常的大规模合并可能导致的那些集成问题。

为了确保集成成功，CI系统通常会在合并新的变更后自动运行一系列测试。提交和合并这些变更时，测试会自动开始运行，而无需人们费心去记住必须运行这些测试，一项活动需要人们的投入越多，完成这个活动的可能性就越小，尤其是当人们很忙时。这些测试的结果通常会以可视化方式展示，“绿色”表示测试通过，认为新集成的构建版本是干净的，而测试失败或“红色”意味着这个构建有问题，需要修正。采用这种工作流，可以更快地找出和修正问题。

## 持续交付

持续交付（Continuous delivery, CD）是一组通用的软件工程原则，允许通过使用自动化测试和持续集成频繁地发布新软件。持续交付与持续集成紧密相关，通常认为是将持续集成更进一步，不只是确保可以集成新变更而不会导致回归到自动化测试，持续交付还意味着可以部署这些变更。

## 持续部署

持续部署（Continuous deployment, 也称为CD）是通过定义测试和验证来最小化风

险，从而将变更部署到生产环境的过程。持续交付确保可以部署新变更，持续部署则表示将这些变更真正部署到生产环境中。

如果软件变更能够更快地进入生产环境，人们就能更快地看到他们的工作成果。能够看到自己的工作成效会提高人们的工作满意度，工作的整体愉悦感会增强，从而带来更高的绩效。这还会提供机会让人们能够更快地学习。如果某个设计或特性存在根本性的问题，人们可以在更近的工作环境中更容易地进行分析和调整。

持续部署还能将产品更快地交付给客户，这可能意味着会提高客户满意度（不过，需要指出，这不是万能的。如果更新没有解决客户的任何问题，那么客户并不会因为得到这个更新的产品而高兴）。这可能还意味着能够更快地验证产品的成功或失败，允许团队和组织根据需要更快地迭代和调整。



持续交付和持续部署之间有什么差别？这是持续交付和持续部署得到广泛使用以来人们经常讨论的一个问题。Jez Humble（《持续交付》的作者）将持续交付定义为一组通用的原则，可以应用于任何软件开发项目，包括物联网（Internet of Things, IoT）和嵌入式软件，持续部署则特定于Web软件。关于这两个概念之间的区别，更多信息请参见这一章的其他相关资源。

## 最小化可行产品

近年来有一个越来越明显的趋势，就是要减少与创建产品相关的开发成本和浪费。如果一个组织花费数年来向市场推出一个新产品，结果发现这个新产品并不满足新老客户的需要，这将是时间、精力和金钱的巨大浪费。

最小化可行产品（Minimum Viable Product, MVP）的思想是用最小的投入创建所提议产品的一个原型，以确定这个提议是否是一个好想法。并不是100%地完成所有开发后再把产品交给用户，MVP的目标是大大减少所做的开发，这样一来，如果需要做重大的改变，之前花费的时间和精力不会太多。这可能意味着缩减特性或高级设置以强调核心概念，或者重点关注特性而不是设计或性能。与精益和持续交付等想法类似，MVP可以让组织更快地迭代和改进，同时减少成本和浪费。

## 基础设施概念

所有计算机软件都要在某种基础设施上运行，这可能是一个组织拥有和自行管理的硬件，或是其他人管理和维护的租用设备，或者是按需使用的计算资源（可以根据需要很容易地扩展或缩减）。这些概念原先只属于运维工程师的工作范畴，而现在由于开



发和运维之间的界限变得越来越模糊，作为与软件产品相关的任何人来说，这些概念对于理解这种环境都非常重要。

## 配置管理

从20世纪50年代起，美国国防部开始将配置管理（Configuration Management, CM）作为一个技术管理准则，现在很多行业中都采用了配置管理。配置管理是建立和维护某个产品的功能和物理特性及其整个生命周期中性能的过程。这包括实现这种一致的性能、功能和特性体系所需的策略、流程、文档和工具。

在软件工程行业中，很多组织和标准机构如ITIL（IT基础架构库）、IEEE（电气和电子工程师协会）、ISO（国际标准化组织）和SEI（软件工程协会）都为配置管理提出了自己的一个标准。与其他大众模式一样，这使得行业中对于这个术语的通用定义有一些混乱。

通常这个词会与各种形式的基础设施自动化、版本控制或服务提供结合在一起，这就与其他学科中这个词的使用有所区分。为了确保本书读者有一个共同认识，我们将配置管理定义为在产品的整个生命周期中识别、管理、监控和审计产品的过程，包括所涉及的流程、文档、人员、工具、软件和系统。

## 云计算

云计算通常就称为“云”，这是指一种共享的基于互联网的计算，客户可以根据需要购买和使用各个云提供商提供的共享计算资源。通过采用云计算和存储解决方案，组织不必购买、安装和维护自己的硬件，从而避免相应的开销。

由于很多云解决方案具有高性能、节约成本以及灵活性和方便性等特点，对于既希望最小化成本又希望提高迭代速度的组织，这使得云成为了一个理想的选择。迭代和缩短开发周期时间是建立devops文化的关键要素。



尽管有些人把云看作是devops的同义词，但这并不是普遍适用的。devops的一个重要部分是能够评价和评估不同的工具和流程，从而找出针对你的具体环境最有效的工具（流程），即使不迁移到某个基于云的基础设施也完全可以实现。

## 基础设施自动化

基础设施自动化是一种创建系统的方法，可以减少人们管理系统及其相关服务的负

担，还可以提高为客户提供的服务的质量、准确度和精度。实际上，一般来讲，自动化就是一种减少重复性工作的方法，从而尽量减少错误，并节省操作人员的时间和精力。

例如，相对于在一个组织基础设施中的每一个服务器上手动运行相同的shell命令，系统管理员完全可以把这些命令放在一个shell脚本中，这样就可以一步执行所有这些命令，而不是分成多个小步骤。

## 工件管理

工件是软件开发过程中某个步骤的输出。取决于具体的开发语言，工件可能有不同的形式，包括JAR(Java归档文件)、WAR(Web应用归档文件)、库、资产和应用。工件管理可能很简单，只是一个提供访问控制的Web服务器，允许在你的环境内部进行文件管理，或者可能是一个更为复杂的托管服务，具有多种扩展特性。与源代码的早期版本控制很类似，可以根据你的预算情况采用多种不同方式处理工件管理。

一般地，工件仓库可以作为：

- 完成二进制文件和依赖文件管理的一个中心点。
- 组织与公共仓库之间的一个可配置的代理。
- 推广内部开发软件的一个集成点。

## 容器

开发和运维团队之间一直以来存在的较大痛点之一是：如何足够快地做出改变来支持有效的开发，同时不会影响生产环境和基础设施的稳定性。有一种新技术可以帮助缓解这样一些摩擦，这种想法称为软件容器（Software Containers），软件容器是一种隔离的结构，可以相对独立于底层操作系统或硬件进行开发和部署。

类似于虚拟机，容器提供了一种方法，可以为容器中运行的代码建立沙箱，但与虚拟机不同的是，容器通常开销更小，而且对支持它们的操作系统和硬件的依赖更少。这就使得开发人员可以更容易地在本地环境中的一个容器中开发应用，再把同样的容器部署到生产环境中，这样不仅可以最小化风险和尽可能降低开发开销，同时还可以减少运维工程师的部署工作。

# 文化概念

这一章最后要定义的是一组文化概念。尽管一些软件开发方法论（如敏捷方法）定义了人们开发软件时可以采用的一些交互方法，不过还有一些重要的交互和相关的文化概念需要在这里介绍，因为这些思想还会在本书后面出现。

## 回顾

回顾（retrospective）是项目完成之后进行的项目讨论，考虑的主题包括哪些方面做得好，而哪些方面可以在将来的项目中加以改进。回顾通常会定期进行（如果有必要，还可以频繁进行），经过固定的一段时间之后（例如，每季度）或者项目结束时都可以进行回顾。回顾的一个主要目标是局部学习，也就是说，如何将这个项目的成功和失败应用到将来的类似项目。回顾的风格可能不同，不过通常包括以下讨论主题：

做了什么？

这个项目的范围是什么，最后完成了什么。

哪些方面做得好？

项目的成功做法，有哪些令团队特别骄傲的特性，另外哪些方面还要在将来的项目中使用。

哪些方面做得不好？

哪些方面出了问题，遇到的bug，没有满足最后期限，以及将来项目中要避免的问题。

## 事后分析

回顾有计划性，通常定期进行，与之不同，事后分析（postmortem）则在发生一个计划外的事件或中断后进行，有些情况下，一个事件的结果会让相关的人感到诧异，至少暴露了系统或组织的某个失误。回顾往往出现在项目的最后阶段，而且会提前做相应的计划，事后分析则完全无法预料，在真正发生意外事件之前，不可能预计到将讨论“这个事件”的事后分析。事后分析的目标是组织性学习，采用一种系统、一致的方法进行事后分析有很多好处，可以包括以下主题：

发生了什么？

事件从开始到结束的整个时间表，通常包括通信或系统错误日志。



## 听取报告

事件中涉及的每一个人都发表对这个事件的看法，包括他们在事件中的想法。

## 补救事项

应当对哪些方面做出改变，以提高系统安全性并避免此类事件再次发生。

在devops社区中，非常强调事后分析和回顾应当是无问责的。很可能会有一个问责的事后分析，会找出对某个事件“负责”的人作为批评对象，但这与devops运动的核心也就是强调学习的思想是背离的。

## 无问责

无问责的概念与问责文化的想法正好相反。尽管Sidney Dekker和其他人在多年前就已经讨论过这个主题，但直到John Allspaw发表了关于无问责事后分析的一篇博文后，这种思想才真正受到了人们的关注，其想法是，如果强调的是学习而不是处罚，事件回顾将更有效。

无问责文化不只是让人们摆脱困境，更重要的是可以确保人们很自然地谈到事件的细节，即使他们的行为可能直接导致了一个负面结果，他们也能开诚布公地说明有关情况。只有充分了解事情是如何发生的，才能真正开始学习。

## 组织性学习

学习型组织是一个不断学习并自我改造的组织……学习是一个持续的、战略性的过程，要与工作集成并与工作并行开展。

——Karen E. Watkins和Victoria J. Marsick, Partners for Learning

组织性学习是收集、发展和分享一个组织知识体系的过程。一个学习型组织会让其学习更全面，把学习设定为一个特定目标，并采取可操作的步骤不断增加集体学习。

将组织性学习作为一个目标，这是区分问责文化与无问责文化的一个重要部分，因为问责文化通常更关注处罚而不是学习，而无问责或学习型组织会从过往经历中得到收获（甚至是不好的经历），从中寻找教训和获取知识。学习可以出现在很多不同层次上，包括个人、团队以及组织，不过组织性学习对公司整体有更大的影响，相比之下，实践组织性学习的公司通常会更成功。

## 小结

我们讨论了与软件以及软件底层基础设施的开发、部署和运维有关的多种方法论，还介绍了有关个人和组织如何处理事件和失败并从中学习的文化概念。

这里并没有全面涵盖所有内容，另外将来还可能开发出新的方法论和技术。在未来数年里，开发、部署、运维和学习等基本主题将继续成为这个行业的核心。

# DevOps误区和反模式

## 无问责

介绍devops之类的概念时，除了讨论这个概念是什么，还可以讨论它不是什么，这也很有帮助。这个过程有助于澄清关于devops的一些常见误区或误解。这一章中，我们会提供devops的额外背景，并定义一些常见的反模式。

## 常见的devops误区

这个行业中，关于devops有很多常见的误区。在你的组织里，你的团队可能需要很努力地阐明和体现devops的信念价值。这一节中，我们将分析人们在组织中建立这种共同语言时遇到的一些问题。

## devops只与开发人员和系统管理员有关

尽管devops这个名字是由开发（development或developers）和运维（operations）这两个词组成，但这更应当算是指出了这个运动的起源，而不是它的一个严格定义。虽然devopsdays大会的宣传语是“结合开发和运维的大会”，不过实际上devops的概念和思想涵盖了组织中的所有角色。至于具体涉及哪些团队或个人，以及他们以何种方式参与，对此并没有一个权威的列表，同样地，对于“实现devops”也没有一种万全的方法。

要帮助开发和运维团队更好地沟通和更高效地协同工作，不仅如此，还可以把这种想法应用到整个公司。组织中的所有团队都应当考虑在内（包括安全、QA、支持和法律团队），从而实现最有效的协作。例如，利用法律和销售团队之间的高效devops流程，可以根据一致的销售产品目录自动创建合同。





任意两个或更多团队都可以由devops原则受益，重要的是，不要限制应用这些想法的范围，也不要简单地把一组孤立的团队替换为另一组孤立团队。在第3部分中，我们将讨论让团队有效投入的有关问题。

## devops是一个团队

一般来讲，创建一个专门的“devops团队”并不是最理想的情况。如果只是创建一个名叫devops的团队，或者把一个现有的团队重新命名为devops团队，这既没有必要，也不足以真正创建devops文化。如果你的组织中开发和运维团队相互之间无法沟通，处于这样一种状态时，另外再增加一个团队可能只会带来更多沟通问题；而不是让问题减少。只有解决了底层的根本问题，才能实现显著、持久的改变。

如果看作是一个绿场项目，创建一个单独的团队作为一个新环境可能是有效的，可以在这个环境中启动新的流程和沟通策略。在大型公司里，这通常是一种尝试改变的有用的短期策略，而随着时间推移，这个团队的成员往往会逐步融合回到指定角色的团队中。

在一个创业公司环境中，有一个兼具这两方面功能的团队是可以的，前提是这个团队要作为一个协作的整体共同承担服务的职责和义务，而不是让某一个人长期待命忙得焦头烂额。管理层仍然需要清晰地划分角色和职责，确保随着公司的发展，这个团队能够根据需要相应扩展。

这本书将介绍不同的团队组织方法以及团队间的沟通和协作策略，不过最终要记住重要的一点：对于如何实现devops并没有一种正确或错误的方法，如果你的团队名字里本身就包含devops，而且确实很合适，就没有理由改变。要记住，devops是一种文化、一个过程，要相应地确定你的团队名字和结构。

## devops是一个职位

“devops工程师”职位引发了一场不小的争议。这个职位有多种不同方式的描述，包括：

- devops工程师是一个同时知道如何编写代码的系统管理员。
- devops工程师是一个了解系统管理基本知识的开发人员。
- devops工程师是一个神话般的10×工程师（通常会说他的效率是其他工程师的10倍，不过具体的倍数很难度量，所以往往只是象征性地称为10倍），他可能是

一个全职的系统管理员或全职的开发人员，只拿一份薪水，但工作质量不会有任何缩水。

这种devops工程师的概念完全是不现实的，不仅如此，这也无法很好地扩展。在一个组织的早期发展阶段，可能有必要让开发人员同时部署代码和维护基础设施，但是随着公司的日渐成熟和发展，应当让人们更专注于他们的本职工作，这样才更合适。

专门设有一个devops主管或另外某个职位由专人负责devops的做法通常没有太大意义。devops的核心是一个文化运动，它的思想和原则要在整个组织中加以应用才真正有效。

不过，现在“devops工程师”这个职位屡屡出现在公众面前。根据Puppet的DevOps薪酬调查报告，与普通的系统管理员相比，有“devops”头衔的工程师薪水更高。由于对devops工程师角色有不同的解释，所以不同组织的角色之间存在不平等的待遇，这会带来很不好的“滑坡效应”<sup>译注1</sup>。既然被称作devops工程师就能让薪水立刻上涨1万美元，又何乐而不为呢？



DevOps薪酬调查报告指出一个重要趋势：“更多devops工程师报告每周工作时间超过50小时，系统工程师报告每周工作时间为41~50小时，系统管理员报告每周工作时间少于40小时”。挑选工作时，要确保不是以牺牲个人时间和更高层次的工作倦怠为代价来换取更高的薪水。

## devops只与Web创业公司有关

基于Web的公司的一个主要产品是用户浏览公司网站时看到的Web应用。这些网站的实现通常不依赖与网站交互时存储的信息（即无状态会话）。网站升级可以分阶段完成，或者通过将实际业务流迁移到网站的一个新版本来实现网站升级，这使得持续部署过程相当容易。

很容易看出为什么devops对于类似这样的基于Web的公司很有意义：devops运动可以帮助打破可能阻碍开发和部署的壁垒。如果一个Web公司的流程太慢，要用几个星期才能修正一个输入错误，与这个领域更新、更敏捷的公司相比，它们将无法与之竞争。

---

译注1：滑坡效应是指一旦开始便难以阻止或驾驭的一系列事件或过程，通常会导致更糟糕、更困难的结果。

并不只是Web创业公司可以从改善的协作、亲密性和工具受益。在小的创业公司里，可以更容易地实现团队结构和流程的迭代；大企业通常已经形成一套体制，往往会抗拒快速改变，政府机构则更甚，可能还有一些法令专门限制和阻碍改变。不过，即使这种组织也可以完成改变。在本书后面，我们会分享一些例子来说明甚至企业和政府组织中也可以应用这些文化思想。

## 你需要一个devops证书

文化是devops中很重要的一个部分，那么要如何认证文化呢？没有一个60分钟的考试能够检验你多有效地与其他人沟通、你的公司里团队的协作如何，或者你的组织如何学习。对于需要使用高级技能的特定技术（如特定的软件或硬件）而言，证书是有意义的。如果公司需要某种特定的技术或技能，根据人员是否有相应证书可以大致了解这个人在该领域掌握的知识。

devops没有要求必须使用哪一个技术，也没有万全的解决方案。证书考试考查的知识往往有明确的正确或错误答案，而devops通常并没有这样的答案。对一个公司最适用的技术对另一个公司来说却不一定是最优的；我们没有办法写出对devops普遍正确的问题。devops证书可能只是一个挣钱的机会，或者只是特定于某个开发商的解决方案而被错误地贴上了“devops证书”的标签。

## devops表示由一半的人完成全部工作

有些人有这样一种印象，认为devops就是让一个人身兼软件开发人员和系统管理员，却只拿一个人的薪水。这种认识是不正确的，而且通常是有害的。曾经有一段时间，相当多创业公司都会为员工提供一些福利，如办公室提供一日三餐，配洗衣房，以此鼓励员工在工作上花更多的时间，那时很多工程师发现他们每周都要工作60~80小时，这些误区会让人们更加偏离工作与生活的合理平衡，而越来越过度劳累，这并不是我们这个行业真正需要的。

在非常早的阶段，如果有开发人员对运维有足够的了解，能够很好地处理部署，特别是了解云提供商和其他“服务”，可以处理运维的大量繁重工作，这对创业公司确实很有好处。在这些阶段，每个员工可能必须身兼数职，但是一个组织一旦过了这些阶段，倘若还希望一个人同时扮演两个全职角色，就会导致倦怠。



devops并不能让公司需要的工程师人数减半来省钱。事实上，采用devops可以让组织提高工作的质量和效率，减少中断次数和时长，缩短开发时间，以及提高个人和团队的效率。



## 实现devops有一种“正确方法”（或“错误方法”）

devops实践和原则的早期采用者，尤其是业界知名的那些公司（如Netflix和Etsy），通常被称作是“独角兽”<sup>注1</sup>，他们在实现devops的“正确”道路上已经抢占了这个市场。其他公司迫切希望得到devops文化带来的好处，有时就会努力模仿他们的做法。

即使一个公司展示了他们成功的devops策略，但这并不意味着同样的流程就是所有环境中实现devops的正确方法。如果以货船膜拜（Cargo culting）<sup>注2</sup>的方式在一个环境中实现流程和工具，可能会创建额外的壁垒，还可能导致人们拒绝改变。



devops鼓励批判性地考虑流程、工具和实践。作为一个学习型组织，这需要质疑和迭代流程，而不是作为“唯一的正确方法”或者以往的成功做法全盘接受。

还要当心另一种情况：有些人会说，如果有人没有追随他们的做法，这些人实现devops的方法就是“错误的”。有一点需要反复强调，尽管对devops团队或devops工程师的一些批评确实是中肯的，但也有一些记录表明，一些公司和个人做出批评只是出于他们自身利益的考虑。devops有意地设计为较为宽松，不像Scrum或ITIL那么严格。实现devops最成功的公司都很乐于学习和迭代，会努力找出对他们最有效的工具和流程。

## 实现devops要花X周/月时间

如果一个组织变革需要得到管理层的某种支持，如devops涉及的有关管理层，可能会对变革提出一些问题，其中通常会问到的一个问题就是需要多长时间能够完成。这个问题本身是有毛病的，它认为devops是一个很容易定义或度量的状态，似乎一旦达到这个状态就万事大吉了。

而事实上，devops是一个持续进行的过程，这是一个旅程，而不是目的地。在这个旅程中，有些部分可能有固定的终点（如建立一个配置管理系统，确保公司的所有服务器都由这个系统来管理），但是有些部分则会一直进行，如维护、开发和配置管理的使用。

注1：在devops中，独角兽（unicorn）是作为devops从业者、革新者和早期采用者的一个互联网公司。不要把它与财务上定义的独角兽（市值超过10亿美元的创业公司）相混淆。

注2：货船膜拜描述了这样一种做法：在一个环境中模仿观察到的行为或实现某些工具，而没有充分理解是什么原因或者在什么场合下使用这些策略才获得了成功。

由于devops主要有关于文化，所以更难预测这些改变需要多长时间：人们打破原先割据的习惯而代之以新的协作方式需要多长时间？对此可能无法很容易地做出预测，不过不要因为这一点阻碍你努力完成这些重要的文化变革。

## devops只是关于工具

尽管工具很重要，但devops并不强制或要求使用任何特定的工具。这个误区是导致人们认为devops只适用于创业公司的主要原因，因为企业型公司通常不容易采用新技术。

devops是一种文化运动。在你的环境中，当前使用的工具也是你的文化的一部分。在决定做一个改变之前，显然要找出这个环境中作为现有文化一部分的工具，了解个人使用这些工具的体验，并观察与其他人体验的相似性和不同点。这种检查和评估可以帮助明确需要做哪些改变。

技术会影响你的组织的工作速度和组织结构。如果要对工具做出重大的改变，尽管这对某个个人或团队可能很有意义，但是可能会以组织整体减速作为代价。

本书中讨论的原则并不要求使用某一组特定的工具，这些原则适用于任何技术堆栈。采用devops的公司与使用容器或云提供商的公司之间有很多重叠，不过这并不表示必须使用这些特定的技术，有些公司没有使用这些技术也成功地实现了devops。第四部分会讨论如何评价、选择和有效地实现工具。

## devops只是关于自动化

devops相关工具中的很多创新可以帮助规范理解，消除团队之间的隔阂，并通过自动化提高速度。这些工具能够消除繁琐的重复性任务，已经得到从业者的关注，如基础设施自动化和持续集成。对于这两种情况，自动化正是技术改进的结果。

如果能自动化完成重复性的任务，把原本要完成这些任务的人解放出来，这种自动化就能帮助他更高效地工作。这种情况的好处很明显：如果能自动化服务器构建，系统管理员就能把构建每个服务器所要花费的时间节省下来，去做更有意思或者更有挑战性的工作。不过，如果尝试实现自动化所花费的时间比自动化节省下来的时间还要多，这就不合适了（见图5-1）。

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE  
EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?  
(ACROSS FIVE YEARS)

	HOW OFTEN YOU DO THE TASK					
	50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
6 HOURS				2 MONTHS	2 WEEKS	1 DAY
1 DAY					8 WEEKS	5 DAYS

图5-1：XKCD的这个漫画对花费和节省的时间做了一个比较 (<https://xkcd.com/1205>)。原先图上还附带有以下alt文字：“别忘了，你在这个图中查找节省了多少时间时，这本身也会花费时间。另外别忘了，阅读这个关于时间花费的提示同样会花时间。此外，你还要花时间确定这么做是否有意义。要记住，时间正一分一秒地逝去，包括现在正在流逝的时间”

关于自动化在一个环境中所起的作用，以及人为因素以何种方式影响我们选择自动化的对象和方法，对此人们已经进行了大量讨论。对这些方面的关注体现了跨学科的同质性，如果我们注意其他行业，这些行业可能会为我们自己的行业提供很多可以借鉴的经验教训。

随着系统越来越复杂，另外由于共享服务的出现，组织之间越来越相互依赖，自动化对我们至关重要。不过，如果没有共同的环境或者没有考虑到人的需要，自动化可能会带来未知的额外风险。自动化可以让工作更快地完成，不过要想做到最为有效，还必须增加透明性、协作和理解。

## 航空业的早期自动化

1977年审查航空业时，美国众议院科技委员会发现飞行员座舱自动化是一个首要的安全隐患。航空领域的研究发现，尽管飞行员仍能驾驶飞机，但目前使用的自动化会导致他们的思维能力大幅衰退。飞行员原本不用借助地图显示也能跟踪位置，能果断确定下一步做什么，还能快速找出仪器系统的故障，但由于自动化



的引入，飞行员的这些能力正在逐步丧失。这对我们也是一个警告，在我们的环境中实现自动化时也要注意同样的问题。工具可能会改变我们的行为和我们的思维方式。

2013年7月，韩亚航空公司214航班撞到了旧金山国际机场的防波堤，3人严重受伤。在调查中，美国国家运输安全委员会(NTSB)发现了很多问题，其中就包括由于过度依赖自动化系统，而飞行员对这个系统并不完全了解，以至于未能充分监控飞行速度。

人类的表现存在着不可靠性，为了弥补这一点，自动化控制系统的设计者无意中为新的错误创造了机会，这些错误甚至比他们努力想要避免的问题更为严重。

——James Reason, Managing the Risks of Organizational Accidents

## devops是一种时尚

由于devops不特定于某个技术、工具或流程，所以不太可能过时或者被某种新方法取代。作为一个改进组织有效性和提升员工幸福感的运动，它会得到广泛采纳并变得通用，而不会过气。

devops与ITIL和敏捷等方法论的一个主要区别是，后者有严格的定义，会在不同的时间增加特定的上下文。而devops与之不同，devops运动由个人、团队和组织的故事和想法来定义。这些对话一直在持续，devops是促成发展和改变的流程和想法的不断演进。

在devops社区中，关于devops是否失去了方向有一些讨论。对这个运动持批评态度的人指出，它过多地由负面概念来定义，也就是说，人们总是在讲devops不是什么，而没有指出它是什么（或者根本没有为它提供一个简洁的定义）。他们还认为devops并没有新意，它只是把之前已有的一些想法换个名字，一旦有了下一个名字或新的趋势，就会被弃之高阁。

推动devops运动发展起来的很多基本思想有不同的名字，它们确实已经存在了一段时间，不过devops的精髓在于，它并不是这些部分的简单累加，这一点是全新的，与以往有所不同。人们肯定反对之前的职能壁垒，推荐学习型组织，支持人文系统或者提倡自动化和度量。

devops运动首次将所有这些思想组合在一起，而且其成功是可以度量的。通过掌握和利用有效的devops，你的组织中所用的工具、技术和流程将不断发展和演进。

## devops反模式

这一节中我们将定义更多术语，帮助你熟悉有关概念。不过，与上一章中定义的术语不同，这些概念通常被看作是反模式（anti-patterns），相对于体现devops优势和有效性的那些概念，这些反模式的想法与之背道而驰。

### 问责文化

问责文化（blame culture或blameful culture）是指人们犯错误时会被责备和惩罚的一种文化，这可能发生在个人层面上，也可能在组织层面上。在这种文化中，通常会在事后分析会或回顾会中不当地使用根本原因分析，着力找出最终导致失败或事故的根本原因。如果这个分析指出“根本原因”是某些人的某些行为，就会批评和惩罚这些人，甚至因为他们对事故承担的责任将其开除。如果必须应对外部审计人员，或者按照给定的一组指标有某种自上而下的制度来提高绩效，往往就会滋生这种问责文化。

严重隔离或割据的环境中，由于缺乏透明度，这就为滋生问责文化提供了肥沃的土壤。如果管理层决定要对每个事故找出一个人或一个小组来问责，想要清除这个“坏苹果”，这就会促使个人贡献者推卸自己的责任，努力把责任转嫁到其他人身上。尽管这种情况下的自我保护是可以理解的，但是这不会带来一种开放和协作的文化。不仅如此，人们开始有所保留，不再毫无隐瞒地提供事故的有关信息，特别是关于他们自己的行为不再坦白，会努力让自己免受责罚。

除了对事故的响应，问责文化往往把责备作为改善绩效的一种方法（例如，哪些开发人员在代码基中引入了bug最多，或者哪个IT技术支持人员处理的客户ticket最少），这会让同事之间产生一种敌对的气氛，因为所有人都不希望自己被责备。如果人们过于关注如何避免别人指责自己，他们就不会对学习和协作关注太多。

### 孤岛

一些团队甚至不与同一家公司的其他团队分享知识，部门或组织“孤岛”描述的就是这些团队的这种思维。孤岛式团队没有共同的目标或责任，他们的角色完全隔离。如果再结合问责文化，这会导致团队间出现信息围篱，以此作为工作保护的一种手段（“如果只有我知道如何完成X，他们就不能把我解雇”），涉及多个团队的工作很难完成或者进展很慢，另外由于这些团队或“孤岛”开始视彼此为对手，这还会导致士气低落。

在一个孤岛式环境中，通常会看到不同团队使用完全不同的工具或流程来完成类似的

任务，人们必须逐级经过管理链条上的多个层次，才能从另一个团队那里得到所需要的资源或信息，而且“推卸责任”的情况比比皆是，也就是把责备、责任或工作转嫁给另一个团队。

对于组织孤岛的做法以及所带来的问题，需要花大量时间和精力来破除和修正，此外还需要做出文化改变。由于软件开发人员和系统管理员或运维工程师自成孤岛，人们希望在软件开发过程中解决这个问题带来的问题，这正是推动devops运动的很重要的一个原因，不过需要指出重要的一点，这并不是全部，组织中还可能存在其他孤岛。跨职能团队通常被称为“非孤岛团队”，不过并不是只有这两种选择，即使一个团队只提供一个职能，也并不表示它一定是一个孤岛。孤岛产生于团队之间缺乏沟通和协作，而不是由于职责的分离。

## 根本原因分析

根本原因分析（Root cause analysis, RCA）方法是要明确事件或几近错失<sup>译注2</sup>的决定性“根本”原因，并采取适当的措施避免问题再次发生。这是一个迭代的过程，会一直持续，直到找出所有组织要素，或者直到已经处理完所有数据。组织要素是在系统生命周期的所有阶段中对系统施加控制的任何实体，包含（但不限于）设计、开发、测试、维护、运维和停用阶段。

识别根本原因的一种方法称为“5个为什么”（5 Why，也称为五问法）。这种方法需要不断地问“为什么”，直到找出根本原因。这要求回答“为什么”的人有足够的数​​据来适当地回答问题。第二种方法是创建一个鱼骨图，这也是更系统的一种方法。Kaoru Ishikawa于1968年发明了这种方法，这种因果图可以帮助团队以可视化的方式分析原因，并将原因归类，找出差异来源，发现不同来源之间的关系，从而对过程行为有深层次的了解。

RCA一般只是要确定一个根本原因。提供事件管理的工具通常只允许一个职责分工。这就限制了RCA的作用，因为它会重点关注直接原因，而没有考虑可能是决定要素的其他方面。对于根本原因分析有一种隐含的假设，认为系统会以一种线性的方式失败（或成功），但任何足够复杂的系统都不是这样。

---

译注2：几近错失是指由于及时的介入，而使其原本可能导致意外、伤害或疾病的事件并未真正发生。



## 人为错误

人为错误 (Human error) 是指一个人可能会犯错误直接导致一个失败,这通常会在根本原因分析中被指为根本原因。这里隐含地认为别人不会犯这个错误,在问责文化中这很常见,必须有人要为事故中所承担的责任接受惩罚。同样的,这个观点也过于简化了,而且会过早地停止调查。这里通常假设人为错误只是由于疏忽、疲劳或能力不足造成的,而没有调查是哪些综合因素导致这个人做出这样的决策或采取了之前的行动。

在问责文化中,一旦找到某个人犯了错误,讨论就会结束,重点往往是谁犯了错误以及最终的结果。在一种无问责文化或一个学习型组织中,人为错误则被看作是一个起点而不是终点,人们会对做出决策的上下文以及当时为什么做这样的决策展开讨论。

## 小结

熟悉这些术语会使你对这本书的其余内容有更深入的了解。结合上一章介绍的基本术语和概念,以及第3章介绍历史时谈到的模式和主题,现在你对这个行业中devops的全貌应该已经有了一个比较清晰的认识。在这些理解的基础上,接下来可以定义和讨论有效实现devops的4大支柱了。

# 有效实现devops的4大支柱

Patrick Debois说过，devops是一个人类问题，这说明每个组织都有该组织中的人特有的一种devops文化。尽管并没有一种能同样用于每一个组织的实现devops的“正确”方法，不过我们发现确实存在4个共同的主题，任何希望实现devops的团队或组织都需要在这些方面投入时间和资源。

有效实现devops的4大支柱包括：

- 协作。
- 亲密性。
- 工具。
- 规模化。

这4个支柱的组合可以帮助你解决组织中文化和技术方面的问题。你的组织尝试改变时，可能一次只关注其中一个或两个支柱，这是可以的，不过最终只有充分组合这4大支柱，才能真正实现持续而有效的改变。

有一点很重要，不要忽略前两个支柱而直接跳过去读有关工具的内容，这两个支柱涵盖了文化和人际交往的规范和价值。要实现成功的devops变革，有效的工具使用当然是必要的，但这还不够，否则我们只提供Chef或Docker的一个最佳实践列表就万事大吉了。实际上，解决组织中出现的人际冲突和团队之间的冲突对于培养持久的关系并最终形成一个devops环境至关重要。

## 协作

协作是通过支持多人的交互和输入来构建一个特定结果的过程。推动devops运动的一个指导原则就是软件开发与运维团队的协作。一个团队与另一个有不同关注点的团队成功合作之前，首先团队内部的个人需要能够彼此合作。如果在个人层次或团队内部都不能很好地协作，这样的团队也不太可能与其他团队很好地协作。

## 亲密性

除了发展和维护个人之间的协作关系，组织内部以及整个行业的团队和部门之间也需要建立紧密的关系。建立亲密性过程就是要建立团队之间的这种关系，不仅要实现不同的目标或标准，同时还要谨记共同的组织目标，以及在不同的人群中培养同理心并促进学习。组织之间也可以应用亲密性，促使公司分享故事并相互学习，从而在行业中建立一个协作的文化和技术知识体系。

## 工具

工具是一个加速器，可以基于当前的文化和方向推动变化。可以把工具选择看作是成功的捷径。需要了解这些工具为什么能带来成功以及它们对现有结构的影响，这对于防止团队和组织掩盖问题很重要。如果没能很好地分析价值、标准和组织结构中的问题，这会导致随着文化负债的累积产生不可见的失败条件。如果工具（或者由于缺少工具）阻碍了个人或团队合作，就无法成功地实现你的目标。如果协作的成本很高，对工具投入不够（或者更糟糕的，把资源浪费在错误的工具投入上）还将增加这个成本。

## 规模化

规模化强调的是组织在整个生命周期中所采用的过程和关键点。除了考虑规模化对于大型企业组织中实现devops意味着什么，还要考虑在组织发展、壮大甚至收缩时如何应用有效devops的其他几个支柱。不同规模的组织运行存在不同的问题，这包括技术和文化两方面，我们就是要分析不属于“一般”小型devops文化的组织的问题。



## 小结

## 第二部分

综合在一起，基于有效实现devops的这4大支柱，可以解决影响软件开发的文化和技术问题。在这本书接下来的四个部分中，我们将深入地介绍这几大支柱。我们找出了这个行业的一些例子，涵盖了从Web创业公司到大型企业等各类不同的公司。尽管没有严格要求按顺序阅读后面的每一章，不过建议你读完所有章节，因为这4大支柱相互组合和融合才能真正保证有效实现devops。

# 协作：个人协作

人们每周在一起工作几小时，彼此之间建立一种持续长久的关系，这非常重要。协作是经过多人的交互、输入和支持来构建一个特定结果的过程。相对协作最早在敏捷软件开发中引入的一个技术，即两个人同时完成同一段代码。这就是协作的一个例子，但这绝不是唯一的协作例子。

## Sparkle公司的每周计划会议

“我真的认为这是一个大好的机会，可以在我们的新评论服务中使用MongoDB。我看过这个教程，非常棒，展示了可以很快、很容易地建立和运行这个数据库，而且不会有其他解决方案的管理开销”。Geordie说，他是Sparkle公司的一位高级开发人员。

General认真听取了Geordie的热情推荐，记下了在Sparkle公司技术堆栈中引入MongoDB可能带来的好处。“有没有人对使用MongoDB有想法或者问题？”他询问开发团队。

“基于目前的技术堆栈，我们已经支持MySQL和它的所有依赖项，我们已经在数据库方面我们已经做了大量工作，增加MongoDB会对我们带来额外的复杂性。与已经使用的工具相比，MongoDB需要来修改数据库，我们不必修改数据库”。这是Sparkle公司的一位高级开发人员。

团队中总是会出现这种意见不一致。正如我们之前所讨论的，团队中建立信任关系，也可能危害这种关系。团队如何管理这种关系，以及他们的组织中协作如何增强或削弱人们的参与。

## 协作：个人合作

人们每周在一起工作几小时，彼此之间建立一种持续长久的关系，这非常重要。协作是通过多人的交互、输入和支持来构建一个特定结果的过程。结对编程是最早在敏捷软件开发中引入的一个技术，即两个人同时完成同一段代码，这就是协作的一个例子，但这绝不是唯一的协作例子。

### Sparkle公司的每周计划会议

“我真的认为这是一个大好的机会，可以在我们的新评论服务中使用MongoDB。我看过这个教程，非常棒，展示了可以很快、很容易地建立和运行这个数据库，而且不会有其他解决方案的管理开销”，Geordie说，他是Sparkle公司的一位前端开发人员。

General认真听取了Geordie的热情推荐，记下了在Sparkle公司技术堆栈中引入MongoDB可能带来的好处。“有没有人对使用MongoDB有想法或者问题？”她询问开发团队。

“基于目前的技术堆栈，我们已经支持MySQL和它的所有依赖库。在MySQL集成方面我们已经做了大量工作。增加MongoDB会对支持和维护增加额外的开销。与现在使用的工具相比，MongoDB能带来哪些改进可以抵消这些开销呢？”Alice问道，她是Sparkle公司的一位高级开发人员。

团队中总是会出现这种意见不一致。人们相互之间以何种方式做出响应，这可能帮助建立后续的关系，也可能会危害这种关系。下面将深入分析devops组合，了解在我们的组织中协作如何增强或削弱人们的关系。



## 定义协作

作为devops的一大支柱，协作是指个人为实现共同目的进行的有意的活动。具体的协作例子包括：

- 异步代码审查。
- 建立文档。
- 更新问题和bug报告。
- 展示每周进展。
- 定期状态更新。
- 结对。



要认识到不同形式协作的价值和作用，这很重要。有些协作工作需要与其他人协调完成，每个人负责协作工作的某一部分，集中精力完成这一部分，从而实现最终的共同目标。还有一些协作工作要连续完成，由两个人或更多的人一同工作来实现一个目标。取决于具体的工作和周围的环境，这些协作方法都是正确的。

如果说某一种协作工作比另一种更好，就好像在告诉所有人跑步是唯一成功的锻炼方法一样。

2015年1月，根据对团队的分析，Anita Woolley和同事在《纽约时报》上发表了名为“Why Some Teams Are Smarter Than Others”的文章，介绍了他们的发现。Woolley谈到一些更出色的团队之所以在表现上优于其他团队，主要是有以下特点：

- 沟通。
- 平等参与。
- 心理理论。

换句话说，有效的协作包括沟通、平等参与和心理理论（Theory of Mind, ToM）。心理理论是指能够识别一个人自己的观点，另外了解其他人根据他们自己的上下文会有完全不同的观点。需要分析个人之间的差异，并研究这些差异会如何影响他们的观点，这会帮助扩展我们自己的心理理论，建立相互理解，并帮助解决冲突，这对于devops组合至关重要，实际上提升了团队成员的能力，使他们更出色。

# 个体差异和背景

我们每个人都有不同的文化背景，有自己特有的经历，这些会影响我们选择如何工作以及为什么工作。尊重个体差异可以帮助建立相互理解，并以某种方式解决冲突，这对于devops组合至关重要。多样化的团队在创造力、解决问题以及生产效率等方面有很多好处，但是这种差异也会导致短期的人际冲突，可能是个人冲突，也可能是专业冲突。

## 专业背景

人们的专业背景各不相同，即任现职之前的相关工作经历有所不同。尽管通常可能认为我们只根据人们当前的工作对他们做出评判，但事实上，人们的背景会以多种方式影响我们的想法、交流和协作。

## 企业经历与创业公司经历

专业背景的一个区别是人们之前任职公司的规模。在创业公司，往往更愿意聘用之前有创业公司经历的人与他们共事。这在某种意义上是有道理的，特别是还处在早期阶段的创业公司，如果核心人员之前有过成功的创业经历，通常就更有可能取得成功，不过有一点要当心，不要对主要在大组织工作（或者只在大企业中工作）的人有过多的偏见。即使只有在大企业工作的经历，也不能说明这个人无法胜任小公司的工作，并不是大企业里的每个人都是“恐龙”，要避免这种偏见或歧视。

这里的难点是，大型团队中任务可能过于细化，而小团队可能有太多工作而导致大量任务切换，需要平衡这二者，而不是完全将有企业经验的人排除在外。

## 技术影响

技术和非技术背景可能导致人们之间产生摩擦。这可能出现在整个公司层面上，一般会认为工程师对公司更有价值，而支持、销售和营销团队通常被当作“二等公民”。如果这种感觉在管理层也有体现，如在一个处在早期阶段的创业公司里，如果共同创始人都有工程领域的背景，这会让非工程领域的员工士气低落。人们需要感受到他们的工作得到认可。

## 角色等级

当然这不仅限于非技术角色。在很多传统的软件开发工作室里，IT和相关角色（系统和网络管理员、运维工程师、QA工程师和数据库管理员等）通常会受到类似的对

待。在核算中，运维往往会归为一个成本中心。为支持整个组织产生的所有成本都被看作是运维带来的负债，而不是考虑这个团队或部门带给组织的价值。

更进一步，运维为组织提供的价值是不可见的，其影响只有在出了问题或者服务中断或降级时才会被注意到。运维团队常常被其他团队看作是阻碍力量，这些问题正是最初促使devops运动发展的一个主要原因。

## 不同的工程路线

甚至是在工程师中，人们的背景也千差万别。在过去，软件工程师几乎都有技术背景，可能有某个领域的一个或多个学位，如计算机科学或计算机工程，或者长期从事计算机相关的工作。很多人都是在很小的时候就开始“倒腾”父母的计算机，然后作为一个“天生的”工程师自学成才学会编程。

最近几年，进入开发领域的门槛已经有了显著变化，因为掌握必要职业技能的教学机制已经改变。编码训练营、短期（3~6个月）技能培训计划和教育型聚会为人们进入这个领域开启了新的途径。想改行的人可以经由这些途径从事技术工作，而不必像以往那样先花4年时间和大笔学费攻读学位。

有些训练营专门为技术领域中没有得到充分重视的弱势群体提供安全的学习空间，如女性或有色人种，对于希望提高工程人员多样性的公司而言，这些训练营是很好的资源，可以从中招募员工。不过，有些地方对于有“传统”工程背景的求职者还是有偏见。这些偏见会影响员工如何看待非工程技能的价值，导致团队缺乏软技能，软技能是影响人们联系、相处及共事的能力。

## 经验

职位等级或经验也可能导致团队成员之间的摩擦。招聘人员时，团队可能表现出希望聘用更有经验的求职者或“高级”工程师，因为他们相信更资深的人员能更快地进入状况，并能更早对团队做出贡献。



高级工程师人数很有限，远远少于希望招聘高级工程师的公司。除了要有更长时间的技术工作经验，还需要对初级员工进行指导和培训才能帮助他们成长为高级员工。考察团队中的人员时，除了他们的技术能力，还要考虑是否能有效地加以教导或指导，确保不会因为不合适地使用人才库阻碍你的发展。



## 个人背景

要增加团队的多样性，这意味着要确保个人背景覆盖面更大，如性别、性取向、种族、等级、主要语言、能力和教育水平，对于一个以产品为中心或者由客户支持驱动的组织来说，多样化的个人背景可以引入更多的经验和观点，从而增强组织的能力。

多样化的人员组成对团队、组织以及整个行业都有好处。不过，多样性也会增加摩擦。如果一个团队有一个主导的群体，例如，团队主要由男性组成，倘若聘用一个女性，就需要对他们的预期、流程和可能的行为做出调整，来缓解这种摩擦。

这种调整可能很简单，只需要改变团队成员相互之间的称呼，例如，从“嘿，伙计”“嗨，哥们”或“先生们”改成更中性的问候，比如“嘿，大家好”。类似地，一些笑话可能不是让气氛更轻松，反而会带来紧张，让人不舒服。关于这种调整，另一个例子是调整工作时间以及对工作生活平衡的期望。假设一个团队里都是单身的年轻人，他们习惯于工作一整天，下班后再聚在一起喝酒。如果这个团队聘用了一个单身父亲，他每天从下午4点到8点都不能与大家在一起，就会发现自己无法参与团队的大多数社交和联谊活动。

HR部门要了解多样性的相关问题，这是组织中很重要的一个部分，可以帮助避免个人之间由于这种个体差异产生摩擦。应当鼓励个人贡献者和管理者接受无意识偏见的相关培训，帮助他们研究哪些假定会影 响他们的工作交流。



这些举措的意义在于培养一个安全、尊重和包容的环境。如果缺乏个人安全，员工之间就不太可能相互信任。个人背景往往会带来权力差异，而这可能影响甚至阻碍协商。

还要记住重要的一点，尤其是由于全球化和远程工作的增加，团队成员更有可能来自不同的文化和国家。举例来说，作为一种正式欢迎礼仪，有人可能鞠躬，有人会握手。文化或地区差异可能会以多种方式体现。

## 目标

团队成员会为相同的目标努力，共同分担责任来实现成功，不过团队里每个人可能还有各自不同的职业目标，如果负面地理解这些差别，就会增加摩擦。要知道，这些不同的动因可以帮助团队成员相互之间更好地建立理解和同理心。

- 有些人把他们当前的职位看作是职业发展的一个“进身之阶”，另外一些人可能只把它看作是一份“工作”，可能供养他们的家庭需要这份工作，也可能他们在考虑改行或者还同时在从事其他副业。要适当地分配工作，以反映每个人优先考虑的事项，这有助于避免一些不好的情况，比如在一些重要的项目中，有些人并没有注意力或关注点放在项目上，这就会妨碍项目的顺利进行。
- 很多人希望学习并进一步发展他们的技能，不过不同的人具体情况也有所不同。分配的工作要与个人的学习目标一致，并明确对团队总体目标的影响和价值。
- 有些人更关注个人的工作，另外一些人却认为发展社交网络或者参与更多关注社区的活动更有意义（如提供指导或者在行业大会上发言）。后者认为埋头写代码的工程师很孤傲，或者没有足够的兴趣了解更完整的行业全貌，另一方面，前者更关注编写了多少行代码或者处理了多少个客户ticket，这些人认为强调社区的问题对于完成“真正的”工作没有太大贡献。要明确团队和公司对于不同类型的贡献有怎样的期望，这对尽可能消除不满很有帮助。

## 认知方式

人们处理信息的不同方式也可能导致个人之间产生摩擦，也就是他们的认知方式或认知风格（cognitive style）。这包括：

- 如何认知事物。
- 如何学习和吸收更多信息。
- 内心里如何融入他们的工作、环境和周围的人。

可以把不同的认知方式描述为不同的轴或谱系。尽管下面的列表并不完备，不过这里涵盖了工作环境中可能有的一些主要的认知方式：

### 内向、中向和外向

这个个体维度度量了人们如何补充能量，也就是为“内部电池充电”。内向的人可以通过独处或者在熟悉的小群体里恢复能量，而外向的人要走出去与很多人交流才能“充电”。中向者介于二者之间，取决于具体情况，他们既能独处也可以通过与他人的交互充电。外向的人可能喜欢群体项目或者组织型角色，在这里他们可以与很多人交互，而且有开放空间的工作环境，而内向的人可能更愿意单独完成工作任务，会认为开放空间的工作环境很折磨人。

## 询问者与猜测者

“询问还是猜测”文化最早出自于2007年的一个互联网论坛帖子，其中谈到人们询问其他人时的不同方式。询问者认为大部分事情都可以询问，而且很清楚可能会得到否定的答案，而猜测者往往倾向于更深入地了解情况，避免询问，除非他们非常确定答案是肯定的。



要说明并通过文档明确指出你希望团队成员如何交流，这会帮助他们建立相互理解，并避免产生不满。

## 开始者与结束者

开始者喜欢提出新想法并着手启动，他们会精力十足地开始一个新项目。结束者则喜欢扫尾，修正项目中残余的问题。如果让开始者去做大量结束者的工作，他们会感到厌倦，而如果要求结束者作为开始者，这会让他们感觉力不能及，不知道从哪里开始。

## 分析型、批评型和横向思维

分析型思维强调事实和证据，将复杂的问题分解为更简单的小问题，并去除冗余的信息或无效的选择。横向思维会比较间接地收集信息，发现遗漏的元素，从多个角度审查问题，消除刻板的思维模式。批评型思维会评价和分析信息，并对其反思来形成判断。这可能包括评价论据的逻辑完备性或偏向性，衡量不同的观点或证据来做出决策，或者分析一个结论在逻辑上能否由论据得出或者某个人的推理是否存在漏洞。

## 纯化论者与实用主义者

纯化论者寻求使用绝对最佳的技术解决一个问题，如果这个完美的技术不存在，他们就会创建自己的技术。对纯化论者而言，如果一个项目要求他们的工程原则打折扣或者有所变通，他们会很不乐意。实用主义者则不同，他们更关注实用性，是尝试创建理想的解决方案？还是考虑到当前环境和约束的现实性采用相应的方法？他们会权衡这二者的成本。实用主义者考虑的是如何实现某个事物的“操作化”（operationalize），使它在实际的生产环境中具体运转，这不同于纯化论者的方法，纯化论者更强调技术本身。

要创建和维护一个合适的工作环境，支持有不同认知方式的人，这很重要。要当心办公室制度不能不必要地倾向于某些认知方式，例如，要求员工早上8点必须在位，而



不允许他们通过远程方式参加会议，或者布局规划中只设计了一个很嘈杂的开放式办公室，而没有留出适当的区域让人们能够在需要安静、不被打扰时专心工作。



聘用员工时，要谨记这些不同的认知方式。查看你目前的团队成员属于哪一类，并注意团队中是否存在不平衡。如果团队里的夜猫子比喜欢早起的人多，这可能没有太大问题，不过，如果你的团队中开始者与结束者之间、纯化论者与实用主义者之间没有一个很好的平衡，就可能导致整个团队的效率和质量出现问题。如果你发现需要聘用某个人来解决这种不平衡，一定要在整个聘用过程中记住这一点。

## 获得竞争优势的机会

公司可以投入资金和资源来灌输质量流程的观念，让个人充分了解对组织至关重要的价值观。

### 指导

要注意教导和指导，这很重要，因为一个不提供指导和支持的环境不可能培养出真正成熟的高级工程师。如果组织愿意为初级工程师的成长和发展提供投入，这将成为一个竞争优势，不只是因为这会壮大组织的人才库，还因为对于高水平的工程师来说，如果他们总能感受到组织的支持，而不论他们当前的经验水平如何，他们就更有可能会留在这里。建立一个正式的指导计划会很有好处。

### 赞助

除了指导者，在提倡和为被辅导者（protege）提供指导的组织中，个人还能从赞助者受益。赞助者也会有很大投入，因为这是互利的。赞助者提倡必要的提升，给予可能的好处，扩展被辅导者的自我感知，并提供有价值的高级领导联系途径。被辅导者受到信任，他们可以得到并利用赞助者的馈赠。经济学家Sylvia Ann Hewlett研究了英国和美国的12000位男性和女性，发现对于可度量的职业发展，赞助比指导更重要。组织应当创建一个全面的计划，使员工了解作为一个赞助者需要什么，如何评定一个可能的被辅导者，以及如何找到和评估赞助者，这很重要。

### 教育

有些组织担心提供太多培训会鼓励人们获得新技能然后利用这些新技能另觅高就。在考虑培训和个人成长时，要注意一个更大的风险：如果不提供适当的培训机会，真正想学习的员工会选择离开，而转向那些对此提供支持的公司，留下的

则是没有经过培训或者没有太大学习动力的员工。这样一来，你的组织会有一个不好的名声，让人认为这是一个不太理想的工作场所，不仅如此，这对于提升员工的整体水平也没有任何帮助。

要提供充分的培训，使他们有能力离开，同时还要对他们足够好，使他们不想离开。

——Richard Branson

## 指导

一个成功的正式指导计划要让指导者和被指导者了解各自的目标、角色和职责。健康的指导关系是双向的，使这个关系中的各个参与者都能成长和学习。了解这种关系可以帮助你成为一个合格的指导者，即使你自己不曾有过指导者。

### 高级到初级指导

最传统的指导类型是高级到初级指导，一个高级工程师指导一个初级工程师，这通常是一个正式指导计划的一部分。这很适合充分利用更高级团队成员的经验来帮助提高初级成员的技能。如果高级员工有很好的沟通能力和讲解能力，还有足够的耐心，能帮助其他人真正地学习（一个没有耐心的人可能不擅于教别人，而会直接坐到键盘前自己完成工作），这种方式最为适用。在最好的情况下，初级员工提出的问题可以帮助高级员工重新审视他们之前认定的做法，可能会质疑一个解决方案是否确实是最好的解决方案，而不只是强调“我们一直都是这样做的”。

### 高级到高级指导

高级到高级指导不太常见，即两个高级别的员工相互指导。在这一类情况下，他们可能会分享大量深层次的知识，但是如果两个人作为高级员工在同一家公司已经任职很长时间，他们可能不会对原来的做法有任何质疑，不会像新人看待事物时那样提出新的观点。

### 初级到高级指导

初级到高级指导是一个初级员工指导一个高级员工的过程。如果完成得好，这会增加向所有人学习的重要性。对于特定的技能，我们的水平各有不同。在特定时间我们关注的方面正是我们能力最强的方面。这意味着，很多情况下，一个初级人员在某个方面可能比更高级的人员水平更高。

## 初级到初级指导

最后一种情况，两个初级水平的员工一同工作相互学习时就会发生初级到初级指导。在快速成长的团队里很可能出现这种情况，有可能这个阶段没有高级工程师参与，或者高级人员过于繁忙而无暇顾及。与单独学习相比，同其他人一起能使两个人更快地学习，不过由于没有经验丰富的人告诉他们有哪些好的实践做法，或者在他们遇到困难时提供帮助，这可能会带来一些不好的结果。

## 引入思维模式

思维模式（Mindset）是我们关于自己的个人理念，以及对我们如何实现各种可能性的认知。通过对动机领域的多年研究，Carol S. Dweck博士给出了僵固型思维模式和成长型思维模式的描述。<sup>注1</sup>对于僵固型思维模式（fixed mindset），人们相信才智和能力是先天的、固有的特点，可能在某些方面天生就很擅长，而在其他方面不擅长，这种状态是不可变的。对于成长型思维模式（growth mindset），才智和能力是学习得到的，可以通过努力和实践提升。思维模式会在很大程度上影响人们如何工作、迎接挑战 and 应对失败。

## 培养正确的思维模式

Jason Moser是密歇根州立大学心理学副教授，研究了不同思维模式的神经机制，他在2011年发表了他的研究成果。他指出，在尝试完成任务和犯错误的过程中，有僵固型思维模式的人与有成长型思维模式的人相比，显示出大脑活动更少，这与另一个发现结果是一致的，即成长型思维模式与对错误的适应性响应相关联。改变我们的思维方式实际上会改变大脑如何运转以及对错误如何响应。

研究表明，人们关于自己的能力以及这些能力具体来自哪里有不同的想法，这些思维方式对于他们如何学习和成长会有很重要的影响。成长型思维模式允许人们和组织在所处环境中更快地适应和学习。在实际中，这意味着个人和团队能够对影响生产的事件更快地做出反应，或者在项目生命周期中更快地响应和改变方向，这对整个企业都有好处。

## 僵固型思维模式

僵固型思维模式认为技能和特点都是先天的、静态的，这会让人们认为必须不断地向

---

注1：Carol Dweck, *Mindset: The New Psychology of Success* (New York: Ballantine Books, 2006)。



其他人证明自己。如果有人认为特点是先天的，人要么聪明，要么不聪明，显然他们更希望向自己和身边的人证明他们属于聪明的那一类。

有僵固型思维模式的人把失败看作是一种证据，证明了一个人天生不聪明，没有天份，或者在某个方面不太胜任。为了避免失败和这种自卑感，有僵固型思维模式的人会远远避开他们有可能失败的场合。他们通常不愿意参加要求他们学习新技能的项目。

人们会避免不确定的情况，从而避免失败和不被认可。这意味着，有僵固型思维模式的人不太愿意为工作学习新技能。他们可能还非常喜欢把自己与同伴进行比较（一种很明显的竞争心态），以增强自己的信心，确认自己确实具备某些特点。

## 成长型思维模式

另一方面，成长型思维模式则会更多地考虑个人学习和学习环境。具有成长型思维模式的人认为，他们的技能和知识会随时间改变。如果目前对某个特定领域不了解，他们相信只要有足够的时间、努力、学习和实践，他们就能掌握相关知识。这并不是说每个人都有潜力成为下一个爱因斯坦或居里夫人，而是说几乎每一种技能至少都能得到提升，甚至充分掌握或完善。

对于成长型思维模式，挑战被看作是学习机会，是获得新技能和知识/实践的必由之路，也是提升现有技能的途径。不同于僵固型思维模式，由于没有对失败的恐惧，有成长型思维模式的人会更多地冒险，同时也会更快地成长。失败不会被看作是指示存在某种个人缺陷的一种标志，这只是学习过程中很自然的一部分。

## 关注个人成长

如何培养成长型思维模式呢？通过结合以下6个策略，可以帮助人们准备面对未来的挑战，并在出现变化时有更大的灵活性。

### 了解基本技能

加入一个团队时，首先要了解这个职位以及团队所需的基本技能。即使你在这个行业已经有多年的经验，但你了解成功加入这个特定团队需要哪些必要的技能吗？

想要得到一个职位时，通常我们都认为需要无所不知才能得到这个工作机会。我们会忙于证明自己，让我们的新同事和管理者相信他们做出的决定是正确的，而通常没有花时间来了解这个工作的基本技能是什么，这包括与各个特定环境关联的文化差异。

类似地，如果我们已经在某个职位上工作了很长时间，可能不会考虑这段时间内已经发生哪些改变。要准备额外的支持，关键的一个方面就是要充分了解基本技能。



谁做什么，如何做，以及为什么做？你肯定不希望出现问题时才第一次了解谁有哪些职责及他们如何工作。应当花些时间来了解这些方面，确保你在去“救火”之前已经牢靠地掌握了基本技能。

每个职位都要有的一个基本技能是观察以及被观察的能力。查看其他人如何处理他们的环境中发生的复杂情况，这可能是你所在的同一个团队，也可能是组织中的其他团队，或者是你的组织之外的团队。共同办公和交换项目可以让你直接接触到解决过类似问题的人，他们不会只做“事后诸葛亮”。也就是说，他们可以帮助你更好地找出和解决当前问题，而不是事后才给出建议。



观察时要记录发生了什么，手抄本比笔记本计算机更有帮助。要写下谁在做什么以及为什么这样做。例如，发生一个事件时，经理是事件负责人吗？或者是否由其他人承担这个角色？写下可操作的结果，以便了解以后怎样做会更好。

研究<sup>注2</sup>表明，手写记下来能更好地记住和吸收新内容。

## 发展新技能

在你的环境中，你可能很擅长某些事情。你很清楚当前拥有哪些技能，所以应当找一些之前不知道的事情去做。如果一段时间内一直在做同样的事情，实际上就是在重复同样的学习经历。这会让你逐步丧失学习能力，你必须重新学习这个过程。

学习的新技能不一定与你现在做的事情直接相关，这可能是团队或组织中的一个空白。例如，如果你在运维团队，你支持的应用是存储，就可以研究一些不同的存储算法，如LevelDB，这是Google编写的一个快速键/值仓库。

以此为例，你要花时间研究如何安装LevelDB，还要了解LevelDB的性能特点，在分析你支持的应用的特点时，可以考虑LevelDB的性能特点能提供哪些背景知识和观点。这种学习能提高你的整体水平，使你能更好地运行和管理你的应用。

---

注2：P. A. Mueller and D. M. Oppenheimer, “The Pen Is Mightier Than the Keyboard: Advantages of Longhand Over Laptop Note Taking,” *Psychological Science* 25, no. 6 (2014): 1159–1168.



取决于你的团队规模，另外还要考虑团队要完成多少个项目，学习可能有一定的弹性空间。应当根据需求和实用性来选择要加强哪些技能，还要考虑你的团队规模，另外要考虑这个技能如何用于其他工作以及其他工作中的学习。

学习了一个新内容后，不能就此结束。要把你学到的知识写出来或者讲出来，与别人分享，让其他人也能了解，然后再选择一个新的学习内容。这个行业在飞速变化，每天都在发布新的工具和实践。你不可能在所有方面都是专家，不过起码可以加强学习能力，当你确实要调整职业方向时，就能轻松地接受新的挑战。你还可以分享你的经验，来帮助你身边的人提升。

## 确认你的优点和进步

如何知道什么时候你做得很好？如何确认你已经有充分了解某个主题，可以选择一个新主题，或者将当前的学习上升一个层次？

尽管我们喜欢有外部反馈，可能会得到认可、指导或正式评估等不同形式的反馈，但大多数情况下，不能依赖外部系统来确定和评判我们的进展。大多数环境中的现有系统并不鼓励我们竭尽能力求最好，也不理解对我们最有意义的开发要素是什么。

有一点很重要，要能够正确地度量你的工作表现和效率。现在有很多提供个人反馈的机制，其中一些相对而言更为有用，不过，你应当能够在没有或不考虑外部系统的情况下给出你自己的反馈。要能够诚实地评价自己，认可我们的度量机制，这样我们就能适当地选择自己想要的职业方向。

Yo soy yo y mi circunstancia(我是我自己和我的境遇)。

——Jose Ortega y Gasset

20世纪60年代末，研究个人的编程生产力时，Sackman、Erikson和Grant发现一些工程师比其他工程师更有生产力。后来，这种思想（即更有生产力的工程师）变成一些公司的口号“我们只聘用10×工程师”。

如果有些人一直都以10倍于小组中其他人的生产力工作，这可能是一个信号，说明这个人目前的职位对他已经不再具有挑战性。在某个方面成为专家很让人钦佩，不过不能以停止继续学习、牺牲弹性和灵活性为代价。

很多信号都告诉我们要努力在一项活动中做最棒的。奥尔特加假说认为，一般或普通的科学家对科学的进步有巨大的贡献。相应地我们提出一个推论，认为普通技术人员对技术的进步做出了巨大贡献。我们要赞美那些知名的思想领袖，同样地，也要赞美默默无闻的普通大众，他们提供了故事、一次性工具、流程和文档，这些造就了我



们的行业现状。他们展示出不只是要考虑生产力，更重要的是，由于标准的不断演进，产生了一组通用的工具、平台和接口，从而减少了其他人设计、构建、测试和使用软件的总成本。

## 确保高质量的精深练习

要实际运用你学到的技能。学习实际上会让我们的大脑重建线路，因为我们会创建新的髓鞘质（myelin），这是大脑里的一种白色物质，可以加快和增强我们的神经脉冲。<sup>注3</sup>要确保不仅有足够的练习数量，还要保证足够的练习质量，这才是关键。也就是说，我们要确保不要用错误的方式练习，否则这会强化不正确的机制。辅导或指导的一个好处（不论技能水平如何）就是能够得到外部指导或反馈，帮助指导我们的练习和活动。

每一天，我们的很多工作都可以看作是练习，都在朝着改变结果的最终时刻努力。反复重复同一件事的练习是一种破唱片（broken record）方法。要想真正增长技能，我们需要做不同的事情并不断进步。

非精深练习的第二种方法是自动驾驶（autopilot）。对很多人来说，驾驶汽车多年后，我们会采用自动驾驶，因为随着年龄的增长，我们的能力有可能改变，驾驶会变得越发困难。这里的挑战是要发现环境在变化，而且不能陷于机械的练习而变成职业惯性。

由于这个行业在不断变化，而且技术的影响在以飞快的速度增长，识别这种惯性至关重要。也就是说，当我们停滞不前，或者过于习惯于某个活动（因为它看起来永远不会改变），就说明存在着惯性。我们必须不断地挑战和提升自己，努力学习新技能并增强现有的技能，确保我们保持高质量的工作，并具有对非常规事件做出反应的能力。

## 建立你的工作风格

我们总是有做不完的工作。如果知道工作没完没了，我们通常会离开这样的环境。加入一个环境时，我们能看到要完成的工作，但不一定了解为什么采用现在的做法。不论是开始一个新工作还是继续现在的工作，都不要只是一味地批评现有的东西，这会限制你的格局，要分析哪些做法是成功的，找出这种环境的优点和改进的机会。

---

注3： Alison Pearce Stevens, “Learning Rewires the Brain.” Society for Science, September 3, 2014, <http://bit.ly/learning-rewires>。

有很多不同的个人工作风格。要确定哪些风格适合你，哪些不适合，从而最有效地完成工作，并对你的团队提供最大的价值，这很重要。在这个学习过程中，还可以对所使用的工具和技术建立自己的风格。

要在周围寻找灵感。在观察其他人的过程中，不要害怕尝试他们使用的工具和方法。随着透明性和开源的出现，人们已经养成习惯分享点文件（dotfiles，也就是UNIX系统上以点号开头的配置文件），以及使用其他快捷方式完成工作。

如果你在这个行业中已经有一段时间，可能已经有了自己的风格。可以评价和明确这种风格的各个方面，了解你为什么采用现在的工作方式。要理解重要的一点，你完成任务所采用的方法可能只是你在一段时间后养成的习惯，而它不一定是完成这些工作的最佳方法，这只是你熟悉的方法而已。通过了解你喜欢什么以及为什么喜欢这样做，你就能得到自由和灵活性，可以尝试采用不熟悉的风格。

## 提升团队风格

一旦了解了你自己的风格，接下来可以学习区分你自己的喜好和所在团队的喜好。要把一个群体变成一个真正的团队，很重要的一方面就是要明确如何让所有人保持一致，共同工作。

一个组织如何看待失败以及对失败采取怎样的行为，这些会影响到个人如何形成自己的一套方法。如果团队成员认为他们要对失败负责，就很有可能拒绝改变他们原先已经习惯的流程。

## 思维模式和学习型组织

这种失败观点既适用于组织，也适用于个人。处理一个失败时，问责文化会寻找可能“导致”这个失败的个人，将他们从项目或这个组织中剔除。这通常是因为问责文化会以一种僵固的方式看待失败。在这样的组织中，人们相信如果有人犯了一个错误，这是因为他们不够能干或者不够聪明，而且由于问责文化把这看做是不可变的，所以不会为这个人提供机会来做出改进。这样一来，整个组织往往会停滞不前。人们没有把关注点放在处理失败和从中学习上，而只是强调如何避免失败。

无问责观点则会用更好的方式处理失败，因为它采用一种成长型思维模式，一方面承认错误的发生，另一方面也认为人员和组织都能够学习、成长和改进。团队目前可能对某些方面不太擅长，但是如果人们不断在寻求更好的做法、学习方法以及改进方法，情况就会好转。基于这种对学习、教育和自我改进的关注，可以得到更睿智和更强健的个人和团队。





因此，通过支持更频繁的反馈，并能对当前状况、要达到的目标，以及二者目前是否一致等方面进行沟通，成长型思维模式和学习型组织可以很好地支持 devops 组合。

## 反馈的作用

Dweck 经过多年研究发现，人们往往会接收到一些反馈，这些反馈的性质将是他们发展一种僵固型思维模式还是成长型思维模式的一个关键因素。如果有人某个方面做得很好，而且他们受到的表扬是“干得好，你真聪明”，这里的重点是聪明，这就把他们推向一种僵固型思维模式，会让他们不太愿意接受有挑战性的任务或者可能会质疑其聪明程度的工作。如果与之不同，人们得到的表扬是“干得好，你真努力”，他们就可能把他们的成功与所投入的努力关联起来，而不会关联到某种天生的品质，这会让他们更愿意接受挑战，而且将来遇到挫折之后还会再次尝试。

在反馈和思维模式领域里，原先的研究主要针对学生时代的孩子们，不过这种思想肯定也适用于成年人，即人们收到的反馈的类型会影响其思维模式的形成。思维模式最早可能在一个人的孩童时代形成，不过甚至僵固型思维模式也不是静态的，有人可能在孩童时代形成了一种僵固型思维模式，但仍有可能在成年后形成一种更重视学习的成长型思维模式。

考虑员工成长和绩效时，这会非常重要。有僵固型思维模式的人往往只关心与他们当前能力直接相关的反馈，而剔除向他们指出将来如何改进的那些反馈。另一方面，有成长型思维模式的人则会非常关注可能帮助他们成长的反馈，会把重点放在学习和自我改进上，而不是他们的当前状态。



在员工审查和反馈阶段要谨记这几点。有人做出反馈时，不论是作为一个管理者还是作为个人贡献者，都应当强调员工的努力、行动、所完成的工作和相应想法，重点应当是员工可以做什么而不是他们是什么，这样才能将他们引向一种成长型思维模式。

不论是正面反馈，还是比较负面的反馈，都是如此。例如，下面来比较两种方法：

### 僵固型思维模式方法

显然 Alice 非常聪明，她凭直觉就能了解分布式系统如何工作和交互。不过她在与人沟通方面比较弱，她不是人们需要帮助时要找的那种人。



## 成长型思维模式方法

显然Alice做了很多工作来了解她要处理的分布式系统，关于这些系统如何工作和交互，她掌握了很深入的知识，从中就能看出她的大量努力。我希望她能想办法更有效地分享她的知识，比如可以通过正式的讲座，或者通过不太正式的方式一对一沟通。

对于这个反馈，Alice会做出怎样的响应呢？尽管“擅长分布式系统，需要在人际沟通方面改进”等信息是一致的，但是反馈的性质完全不同。僵固型思维模式的说法（“一个聪明的人”、“凭直觉就能了解”、“不是那种人”）暗含了Alice天生的特点和不可改变的事实。而成长型思维模式的说法（“很多工作”、“大量努力”、“想办法分享她的知识”）强调了Alice的工作和行动，她在过去做了什么以及将来应当做什么。

## 审查和评级

对员工提供反馈时有两个目标。首先，绩效审查等形式的反馈是为了让人们知道他们目前做得怎么样，使他们个人可以成长、提升自己的技能，努力填补知识或技能的空白。其次，除了对个人提供的好处，组织也能明确哪些人表现更好、贡献更大，从而得益。这里的原则是，如果有些人的表现不如他们的同伴，或者一直都无法改进，最好不要留在组织里。

## 反馈频度

根据华尔街日报的文章，51%的公司都会做年度绩效审查，而41%的公司会每季度做一次绩效审查。不过，越来越多的公司逐渐意识到，如果反馈和审查的频度更高，会带来更大的影响，前提是反馈本身对收到反馈的人确实有帮助。显然，如果反馈没有提供新的或可操作的信息，更多的反馈则没有好处。不过，如果反馈确实有用而且可操作，反馈频度越高，对个人和组织的好处就越大。

如果某个人的工作表现不佳，倘若必须等待一年来得到他们下一年的年度审查结果，这对于涉及的任何人都没有好处。在这么长的时间里，他们可能认为自己干得还不错，以至于审查时还会奇怪为什么会得到这样的反馈。反馈心理学显示，对于这种负面的质疑，人们通常会做出情绪化的反应，而不是理性对待，这种现象称为杏仁核劫持或情绪劫持。<sup>注4</sup>因此，人们不太愿意去充分理解他们得到的反馈，也不会对这些反馈采取行动。

---

注4：Daniel Goleman, Emotional Intelligence: Why It Can Matter More Than IQ (New York: Bantam Books, 1996)。



反馈周期更小、更短，意味着调整更小，相应地调整也更容易。这是促使团队从采用瀑布模型完成软件开发转向敏捷方法的一个重要的决定因素，也是为什么持续交付如此有效的一个原因。年度绩效审查类似于瀑布模型，由于得到反馈存在延迟，这会让问题的解决速度减慢，所以组织应当转而采用更敏捷的持续反馈思想。

## 评级系统

特别是在较大的组织中，通常使用不同的评级系统对员工绩效归类或分类。近年来最大的变化之一是从分级评级转向其他评级系统，分级评级也称为被动评级或被动分布。这种做法由20世纪80年代通用公司当时的CEO 杰克·韦尔奇推广开来，它基于这样一种思想，即前20%的员工最有生产力，而中间的70%马马虎虎，只能说可以胜任他们的工作，余下的10%表现不佳，应当开除。这种做法通常称为“评级与封杀”。这种评级系统激励员工不要做后10%。

如果一个系统中要求个人与其他人竞争成就，这会增加有效协作的难度。在个人看来，清晰、透明的沟通没有好处，因为掌握信息可能会影响奖金、职位晋升，甚至还会影响一个人是否能保有工作。分级评级会负面地影响绩效而不是帮助提升绩效，尤其是如果这个过程没有得到充分的解释，它的负面影响就更大，不过庆幸的是，近年来使用这种评级系统的组织在明显减少。<sup>注5</sup>

另一方面，很多创业公司希望避开评级系统，想要完全废除评级和审查。在混乱和变化中（这也是这些初创公司的鲜明特点），缺乏反馈对个人是有害的。另外，由于没有正式的过程或指导原则，很容易出现偏向性，这可能是有意的，也可能是无意的。如果有一个正式的反馈系统，再结合有用而频繁的反馈，就能为希望做出改进和发展职业生涯的人指出清晰明确的前进道路。

考虑这些因素时可以看到，对个人绩效的反馈和评级会影响整个团队和组织的协作，而不只是一次影响某一个人。如果将绩效审查转换为一种零和博弈<sup>译注1</sup>，就会抑制沟通和协作，因为每一个人都会更关注保护他们自己的工作，而不是为公司整体创造价值，更不用说为客户提供最大利益。

---

注5： Max Nisen, “Why Stack Ranking Is a Terrible Way to Motivate Employees,” Business Insider, November 15, 2013, <http://bit.ly/stack-ranking>.

译注1：这是博弈论的一个概念，属于非合作博弈，指参与博弈的各方在严格竞争下，一方的收益必然意味着另一方的损失，博弈各方的收益和损失总和永远为0，双方不存在合作的可能。



反馈的频度和正式程度对于创建协作环境也有很大作用。在这个过程中，比较正式固然很好，不过想想看，相对于大规模的年度审查，每周碰头会上信息流动会容易得多。反馈周期更短，人们会更多地接收反馈并做出回应，就能在两个方向上更好地实现信息分享，而不仅仅是从上到下的信息传递，这样从整体上就能创建一个更加协作的环境。

## 超级明星和超级鸡群的问题

随着诸如“明星开发人员”和“10×工程师”等概念的越发普及，很多公司和招聘经理都在努力招聘那些难得的“超级明星”，希望得到他们预想的10倍生产力。不过，过于关注这些人可能更有害而不是更有利。

国际知名的商界女企业家 Margaret Heffernan在她的2015年TED演讲中，使用了“超级鸡模式”一词来描述公司聘用精英的做法。这是基于普渡大学生物进化学家威廉·缪尔对鸡生产力的研究提出的。

缪尔发现，一个由普通鸡组成的普通鸡群，经过6代的生存繁衍后，生产力会提高。缪尔还建立了一个“超级鸡群”，由生产力最强的鸡组成，而且每一代只选择最有生产力的鸡繁殖下一代。但结果并没有变得更有生产力，这个超级鸡群最后只剩下3只鸡，其余都死了。“超级鸡”只是单个来看更有生产力，而且要以其他鸡的生产力为代价。

在工作中也可以观察到这个结果。麻省理工学院的一项“生产力和创造性解决问题”研究发现，最有生产力和创造力的团队并不是由“超级明星”工程师组建的团队。根本不能根据智商和原先的工程才能预测最好的团队。实际上，研究者发现最好的团队都有较高的社交灵敏度，会给每个人几乎相同的时间发言，而且其中都有更多女性。是不是更多的女性能帮助提高其他人思维的灵敏度，并让人们的发言时间更接近相等（女性通常更有同理心，会更多地倾听，更少打断别人）？我们无从知晓，不过有一点是明确的，提升社交灵敏度，或者能够认知和理解其他人的情绪和一般社会规范，以及沟通，这些是影响团队生产力的决定因素。

## 社交资本对团队的价值

社交资本或人们的社交网络和互动的价值体现在更大的信息流、互惠互利，以及相互依赖和信任。可以与只关注超级明星员工的团队做个比较，在这样一个团队中，可能只在一个方向上提供帮助，而且信息也只在一个方向上流动，这里不存在相互依赖，而且很可能没有多少信任。



需要花时间来发展社交资本，随着时间的推移，它的好处会越来越明显。要想得到我们希望的高生产力的团队和组织，就不要再过份关注消蚀信任和社交资本的“超级明星”员工，而要重视在现有的团队中培养同理心，要努力协作而不是竞争。

## 沟通和冲突解决方式

由于这个行业要求更高的产品性能，同时还要求更低的成本，人们往往发现会面对不一致的需求，它们在时间和关注点方面存在冲突。这些需求带来的冲突要以某种方式加以解决。对此有很多不同的方法，在这里我们将通过冲突解决或协商方式来解读。

谈到协商时，我们是指旨在达成一致的沟通，这可能有多种不同的方式，我们将在下一节逐一说明这些方式。要在一个团队或工作场所中发展协作，使之成为主要的协商方式，归根结底还是需要沟通。我们最早介绍devops组合时已经见过这种想法的实际应用，如果没有有效的沟通，不论是共同的目标、达成这些目标所采取的策略，还是相关的计划，都不太可能成功。

### 有效的沟通

通过有效的沟通，人们可以达成共识并找出共同目标，而不是一味地相互竞争。除了简单地回答问题，或者告诉某个人下一步做什么，还有很多不同的原因可以解释我们为什么要相互沟通。4个主要的原因分别是增加理解、确认影响、给予认可和建立社区。

沟通可以帮助在个人之间和团队之间建立弹性，使人们知道他们不是孤立的个体，能够分享处理策略，并在人们和群体之间传递知识，而且沟通是否有效会对组织整体产生显著的影响。

#### 增加理解

沟通很重要的一部分就是要增加理解，这可能是更清楚地了解某个人期望从我们得到什么，也可能是更深入地掌握一个技术主题的知识，或者介于这二者之间。这种知识可以显式地分享，例如通过研讨会或正式讲座，也可能不那么明显，人们在参与一些活动时（如团队非正式会议或有关bug修复的讨论），会从中了解一些想法、标准和习惯。如果一种文化注重学习和关于知识分享的社交活动，就能提供额外的上下文，增进人们的理解，这是自学无法提供的。



对于自己负责的系统和流程，人们通常会掌握大量上下文知识和情境感知信息。如果没有主动地将这些知识分发给其他人，就会建立知识孤岛，这会很脆弱，很容易受组织的外部事件所影响。如果只有少数几个人了解某个主题的内容，这也会增加这些人的压力（“不，George不能去度假，只有他能修复数据库！”），使他们负担更重，产生倦怠的可能性也更大。可以通过沟通来分享知识并增加理解，这是提高技能同时增强组织健壮性的一种好方法。

很多情况下，理解还包括从历史的角度考虑。对于我们要处理的复杂系统以及它们随时间演进的组织方式，新加入团队或项目的人并不总能明显地看出为什么要采用现在的做法。这种上下文知识对于充分理解系统并做出贡献非常重要。运维团队尤其如此，通常运维团队要确定某个方面是否反常，这个告警是一个假警报，还是确实存在一个问题需要调查？如果能沟通历史的上下文信息，就能让新人或比较初级的团队成员更快地成长，更快地增加知识和增进理解。

## 确认影响

有多种不同的影响方法，相对来讲，其中一些方法更为积极或更有协作性。当然，对于持不同意见的人，可以打断他的发言，也可能讲话嗓门最大或时间最长，或者利用某种权力或高压措施，以此来影响其他人。这些方法都无法得到一个健康或有同理心的团队，尽管确实会有影响，但是其他人会心有不满。后面还会更深入地讨论，要影响其他人，最有效的方法是找到足够的共同点，使他们不仅会做你所想，而且还会想你所想。

## 给予认可

给予认可是人们沟通的另一个常见原因。给予认可可能提高士气，因为人们显然都希望他们的工作和成就受到注意和赞赏。这会增进员工之间的合作，因为员工相互之间更能感受到对方的慷慨和帮助。另外，在工作中这还有助于加强你希望看到的行为。认可通常有两个方面：识别或认识到应当得到认可的方面，以及具体沟通这种感受。

能够找到适当的机会给予认可，这是一项需要花时间培养的技能，如果你的心态不好（例如，你正处于一种负面情绪中，也可能繁重的工作压得你喘不过气，或者你在一个竞争很激烈、“人人为己”的团队环境中），就较难发现什么时候适合给出夸奖或认可。沟通认可也是一种技能。有些人感觉夸奖别人不太自在，特别是如果之前在工作场所里没有给出过太多认可。公开给出认可可能比私下里夸奖更让人不舒服，不过另一方面，如果得到公开夸奖，如在团队会议上受到夸奖，人们感受到的认可度更高。



## 建立社区

最后，通过更好的沟通来加强人与人的关系，可以帮助建立社区。正如之前提到的麻省理工学院的研究中指出，有更高社交灵敏度和更多平等沟通的团队往往更有创造力和生产力。建立社区与这些方面是紧密相关的。如果团队中人们会定期讨论工作相关问题之外的其他主题，会有更高程度的信任和同理心，而且可以更好地保持较高的生产力，并作为一个团体应对压力。如果人们将彼此看作是具体的个体，而不只是抽象的email地址或公司员工目录里的记录，他们在个人层面上往往能更好地互动。

### 实践社区和兴趣社区

实践社区是有相同角色或关注点的人组成的团体，他们会定期见面，希望改进他们在组织里的表现。一个组织中的每一个角色都有机会组成一个实践社区，所以可能有一个开发人员社区，一个QA和测试工程师社区，还可能还有一个流程管理员社区。另外可以围绕特定的工具或语言形成实践社区，不过不论哪一种情况，这些社区都不仅限于某一个项目或团队的人员。如果不受管理层的管制，这些社区可以有机地发展和变化，在这种情况下它们往往运转得最好。与角色和项目一样，社区活动会随时间衰减和流动。需要指出重要的一点，实践社区会关注某个角色，也仅限于积极从事这个角色的人，所以学习和讨论来自人们的实际知识和经验。

兴趣社区与实践社区类似，不过并不只限于从业人员，往往由对组织中的管理、控制和团队沟通感兴趣的人组成。他们可能负责监督或创建实践社区，或者讨论其他更高层次的问题，这些问题对从业人员讨论的当前实际问题可能没有太大的影响。有些社区则以一种不同的方式使用这个术语，将兴趣社区定义为任何有兴趣讨论一个特定主题、团队或技术的人，即使他们自己可能并不亲身实践。实践社区和兴趣社区都是跨职能的团体，其重点在于学习和共同的目标。

不要期望员工在工作之外都成为好朋友，认识一个人和与一个人关系很亲密是不同的，这二者之间有清晰的界限。相对来说，有些人更愿意分享他们个人生活中的点点滴滴，这没有问题。关键不是强制社区建立人际沟通，而是要为沟通创造机会，以柔和的方式鼓励交流，使沟通自然地发生。建立关系和建立社区都要花时间，而不可能一蹴而就，也不能强制。





可以有共同的茶歇时间，留有足够长的午餐时间使人们能边吃边聊，另外有共同兴趣的人可以开展一些选择性加入活动，这些都能很好地促进建立强大的社区。

## 如何沟通

要建立最有效的沟通，你选择的沟通方法要取决于消息的上下文、紧迫性和重要性。这种自我意识使你能更清晰地表达你的想法，增进团队内部的理解，让其他团队和个人能够更容易地与你共事。另外，你要考虑面向什么类型的听众，以及需要他们提供多少上下文信息、有多大投入才能实现有效的沟通。你可能还要考虑沟通的条理性，而不是太过天马行空。

不同的文化会强调不同的表达方式。一些文化强调直截了当，愿意直面反对的观点并在社交互动中直接表达情感。另外一些文化强调间接性，这要求个人从字里行里仔细捉摸，以努力维护团队的和谐。

## 沟通媒介

人们在工作中通常会使用一些方法进行沟通，这样的沟通媒介或方法已经有很多。并不是每一种媒介对所有情况都是最有效的，而且不同的组织和团队偏爱的沟通媒介也不同。表7-1提供了一个并不完备的沟通方法列表，根据一组要素对这些方法做了简单比较。

表7-1：不同沟通工具和方法

	直接性	受众范围	投入	所需上下文	组织
email	低	高	中	高	中
现场（或视频）会议	高	低	中	低	低
聊天	中	中	低	高	低
正式会议	非常低	高	高	低	高
推特或其他微博工具	低	中	低	高	低
GitHub拉取请求	低	中	中	中	中
便利贴	非常低	中	低	高	低
PagerDuty页面	高	高	高	中	低
Nagios告警	中	高	高	中	低
图书或博客文章	非常低	低	中	中	高
图片、图表和GIF	低	低	低	高	低

下面来更详细地分析这个表中的各列是什么意思。

- 直接性 (Immediacy) 是指可以多快地建立沟通, 例如, 走到某人面前直接性就很高, 因为你可以拍拍他们的肩膀, 打断他们, 而email的直接性很低, 因为你无法控制其他人多久查一次email。会议的直接性可能非常低, 因为安排会议可能非常耗费时间, 不仅要考虑人员是否能参加, 另外还要考虑是否有会议场地。
- 受众范围 (Audience reach) 是指一个媒介能否让你将信息传达到所面向的全部受众, 所以如果向一个人发送email, 这个人看到email的可能性就很高, 而聊天消息可能只会被恰好在线 (或者在一个给定频道) 的人看到, 这取决于你的聊天工具的离线消息和提示选项设置。
- 投入 (Investment) 描述了人们参与某种形式的沟通时所需的时间和精力。在投入方面, 会议的投入最高, 因为人们必须从他们的工作中留出时间, 可能要去某个地方开会, 或者远程地参加会议。email、图书或博客文章则需要中等程度的投入, 人们要找时间来阅读, 而聊天或推特等只需要很低的投入。
- 上下文 (Context) 是指一个给定的沟通媒介需要多少上下文信息, 或者如果没有上下文, 出现误解的可能性有多大。推特、聊天和email的上下文需求很高, 因为很容易错误地解读措辞或语气。一般来讲, 文字沟通越简短, 就越有可能因为缺少上下文而产生误解。现场(或视频)形式的沟通上下文需求则较低, 因为人们可以看到肢体语言、听到声音和语调, 而且可以很快地提出问题和澄清误解。
- 组织 (Organization) 是指对于一个特定的媒介, 是否需要为思想或想法加以组织, 即想法的组织性或条理性。会议的组织性需求很高, 因为会议要有一个日程表, 以免浪费人们的时间。email的组织性为中, 因为人们可以在发送email之前稍稍组织他们的想法, 而聊天和推特的组织性为低, 因为它们通常非常快速而且简短。

在第四部分中我们还会介绍更多有关内容, 关于沟通方法, 并没有一种十全十美的解决方案。即使要考虑紧迫性和受众范围等因素, 也不能忽视个人的喜好, 与认知方式一样, 通常人们对于沟通方式也有不同的喜好。有些人非常喜欢面对面的沟通, 这样他们可以从肢体语言和面部表情收集到额外的上下文信息, 而另外一些人可能更倾向于书面沟通, 因为这样他们可以更容易地获得信息。

## 协商或冲突解决方式

除了沟通工具，沟通还包括我们的协商或冲突解决方式。协商时，可能会看到以下不同的协商方式：

- 竞争（Competition），一个人只关注他自己的需求，而以其他人的需求为代价。可能是某个团队成员只为自己选择“好”项目（如会得到大量好评和认可的项目），而把不太讨喜的麻烦工作交给其他团队成员。
- 迁让（Accommodation），一个人接受其他人的需求，而以牺牲自己的需求为代价，有时他的目的是通过帮助别人来与他们建立更好的关系。例如，一个员工可能决定支持某个人的想法，希望这个人会记住这一点，将来能有所回报。
- 回避（Avoidance），双方都努力避免直接冲突，通常会带来消极对抗行为和间接冲突的增加，使关系更为紧张。在这样一些情况下，可能会看到很长的电子邮件线索（email threads），人们都想方设法推卸工作或责任而不直接指出，或者一个团队内部可能对另一个团队很有意见，但从不与被抱怨的对象直接交流。
- 妥协（Compromise），双方都努力找出一种彼此都认可的解决方案，这要求所有人都对自己的需求有所让步，从而得到一个“公正合理”的结果。各方对于给定版本中最后要加入哪些特性或者对于具体的最后期限往往有不同的需求，妥协方式会让人们有一个中间立场，或者对于一个人在另一个团队的特定项目上可以投入多少时间达成一致意见。
- 合作（Collaboration），这与妥协类似，也是一种双赢的策略，不过需要更多的相互理解和学习，并确认各方的需求确实得到了满足。在合作环境中，可能会看到来自不同团队的人结对完成某个特定的项目或特性，产品部署和维护所涉及的不同人员会共同分担待命职责，或者由项目规划中涉及的多个团队的人员共同分担这个职责。

一个团队要想以最好的方式达到其目标，这个团队的成员必须能够协同工作。协作性更强的团队整体来讲更有生产力，而且成员也更热爱这样的团队。另外由于更少出现倦怠，这更有利于提高团队士气和生产力，这是一个不断增强的良性循环。

## 沟通上下文和权力差异

我们要以何种方式沟通，这在很大程度上还要受沟通上下文的影响。如前所述，这不只包括各种沟通媒介和方法提供的上下文信息，还包括要进行沟通的情况和场合。



## 上下文和地点

每天工作中的正常沟通与发生紧急情况时的沟通往往有所不同，如网站中断或出现其他运维问题时。在正常的工作交流过程中，分享笑话、表情包和有趣的小猫图片是建立友情和信任的一种很好的方法，但在出现问题期间，这些可能反而让人心烦。有些公司非常依赖于员工之间的聊天，他们有一些很好的做法，会创建一个单独的聊天室或专门为这些情况建立交流渠道，如预备一个“作战室”方便人们针对主题进行直接沟通。这也有助于团队在之后审查沟通情况，如完成事后分析。

最后，团队成员是否同在一个地方也对他们的沟通有很大影响。如果一个公司只是刚刚开始允许人员远程工作，或者根本不关注或重视他们的远程员工，远程通信和合作就会受到严重影响。如果与工作相关的大部分决策都是当面做出，通常都在一种正式会议场合下做出决策，远程工作的员工就会发现自己可能遗漏了一些重要的的信息或讨论。



处理这个问题的一种方法是把沟通“默认为是远程的”。这意味着，要尽可能多地采用支持远程的沟通方法，最常见的是email和群聊，并把这些方法作为首选方法，而不是最后不得已而为之的做法。不必走到一个同事办公桌前去问问题，如果允许员工在团队的聊天室里提问题，远程工作的员工就有机会学习和参加之前他们无法参与的讨论。这会获得更多信息，整个团队也会有更好的可见性，正是出于这个原因，对于不太敏感或者可能普遍关注的问题通常更倾向于采用群聊方式讨论，而不是一对一或私聊。最后，要有一个可搜索的沟通记录，以便将来参考，这也很有意义。

## 权力差异

权力差异也会影响沟通的上下文，一个组织中出现权力差异的原因有很多。可能简单的只是组织本身的权力结构，管理者比其下属权力更大，或者高级工程师比初级工程师的权力更多。不过，有时权力差异可能比较微妙，相对于受重视的群体或主导群体的成员，技术行业中少数群体的成员权力会比较小。

这些权力差别可能对人们之间的协商方式产生显著的影响。一方面，有较小权力的人可能会避免任何形式的冲突，不希望被利用或者必须做出他们不想做的让步，因为他们知道他们的权力小，往往只能是他们“妥协”，而实际上这根本不是一个双方同时让步的妥协。另一方面，有较大权力的一方也会避免冲突或协商，因为他们认为没有这个必要：如果可以将他们的意愿或解决方案强加给其他人，那些人只能接受而没有其他选择，他们又何必去协商呢？考虑沟通和冲突解决时一定要谨记这些权力差别，这非常重要。



## devops反模式：沟通和打断

尽管不能简单地说“如果你做了X，就说明你在实现devops”，但通过观察反模式（具体来说就是“如果你做了Y，则意味着你没有真正做到devops”），通常可以明确devops的关键要素。在这本书中，我们会在适当的时候展示这样一些反模式，使用示例或一些小案例来说明为什么有些行为或做法并不是有效的做法。

在某些组织里，人们相互打断和插话的情况极为常见。观察你参加的会议，数一数人们发言多长时间就会被别人打断，另外谁最喜欢打断别人。在极端的情况下，你可能会发现，人们几乎总是强行打断别人的讲话来发言，这就导致大量时间都浪费在重复之前别人讲话时已经讲过的内容，而且为了让别人能够听到，人们说话的声音会越来越大。

这种中断文化是竞争型沟通而不是协作型沟通的一个例子，这种沟通往往不会在个人和团队之间建立信任，其目的是为了影响别人，而很少去理解，在社交资本方面不会有多少收获。减少打断行为不仅可以增进理解，还可以帮助人们建立认同感，感受到别人在听他们所讲，这会进一步增加信任和同理心。

例如，有研究表明，女性以男性同样的方式表述事情时，会被认为更“苛刻”、“粗暴”或者“过分”，而男性则被夸赞为“直接”和“负责任”。另一方面，如果女性通过道歉或者使用类似“我只是……”之类模糊的说法，又通常会得到负面评价。如果她们像男同事那样打断别人，往往被认为“不受欢迎”，但是如果不打断别人，在这种办公室文化氛围中，可能又很难让别人听到她们的观点。这些环境对我们的沟通是否成功会有非常大的影响，一定要牢记在心，特别是希望提高团队的多样性和包容性时更要注意这一点。

## 同理心和信任

有效的沟通除了对信息发布至关重要，这也是个人之间建立同理心和信任基础的关键，正是基于这个基础，devops才能有效工作。这又回到本书前面介绍过的devops组合的根基。为了继续朝着共同的目标前进，我们需要建立同理心和相互信任。

要创建这种devops组合，需要能够建立和达成一个共同的愿景或目标，这是所有团队努力结果的共同之处，尽管细节可能有所不同。除了增加同理心和明确共同的关注点，共同的愿景会让个人更清晰地了解组织全貌或更宽泛的目标，这会帮助指导个人的自主行为。如果目标过于模糊或看起来不太相关，充分理解和领会就会比较困难，人们可能没有足够的动机、上下文或能力来选择有效的行动。



## 培养同理心

同理心是理解和分享其他人情感的能力，这是一种可以学习的技能，而且很有必要学习和培养。这种技能的好处已经越来越为人所知，不仅在工作场所中很重要，在工作场所以外也同样重要。更有同理心的人往往不会以个人为中心，不会太过强硬，而且不会那么刻板地看待其他人。有争论或意见不一致时，他们更有可能妥协，而不是采用之前讨论的其他协商方式。

尽管同理心大多是在孩童时期培养，不过还可以采用很多方法在成年后学习和增加同理心。我们将介绍最常见也最有效的一些方法，并说明如何在工作场所应用这些方法。

### 倾听

倾听对于建立一般的同理心很重要，不过在意见不一致或者出现白热化的讨论时，倾听的意义会更为突出。与别人意见不一致时，我们通常只是等着轮到自己发言，思量着自己要说什么，而没有仔细倾听别人在说什么并努力理解他为什么那么说。试着慢下来，要求自己倾听，而不是打断别人，等别人讲话结束时再考虑你自己的回应。

在这里，另一个很好的技能是主动倾听。这包括向讲话者做出反馈，指出你对他所讲的内容有何想法，复述或给出总结，确保你听到的和所理解的确实是他想要表达的，从而确保双方的想法一致。



要注意非语言暗示，如音调、语速、肢体语言和面部表情，这也是倾听的一个重要部分。由于这些非语言暗示不能通过文字传达，所以要有很好的视频（或者至少有音频）设备，如果你有远程工作的员工，这就很关键。

### 问问题

除了倾听，问问题也是建立理解和明确含义的一个很好的方法，这通常是主动倾听的一部分。除了向其他人问问题，我们还可以向自己问问题。要培养对其他人的好奇心，不论他们是陌生人还是其他团队的成员，这个习惯可以帮助我们理解其他人的想法，增强我们的同理心。

这些问题可能是“说到某某，你能不能再说明一下这是什么意思？”，也可以问更假设性的问题，如“怎么考虑火车上的那个人现在是要去哪里？他们在手机上看什么？”，或者是问自己的一些问题，如“我的哪些潜意识的偏见会影响我的观



点？”。要倾听从我们自己或其他人得到的答案，与之结合，问问题是建立同理心的一个非常强大的工具。

## 想象其他看法

除了询问其他人可能在想什么、做什么或者有什么感觉等假设性的问题，我们还可以尝试让自己设身处地地考虑其他人的想法。一方面，我们要假设他人的意图是好的，不过还可以走得更远，问问自己，我与其他人意见不一致时，他现在可能有什么感觉？他有哪些好的意图？可能有哪些正面的动机，这对我们的讨论（或意见不一致）有什么影响？他们反对我的观点有哪些合理的依据？

## 接受个人差别

除了想象其他人的想法、观点和动机，我们还可以让自己接受这些差别，这也是培养同理心的一种方法。与他人一起工作，认真地倾听，把自己想象成与我们共事的其他人，这些可以帮助我们破除偏见（包括有意和无意的偏见）。

考虑这一章前面介绍的不同工作方式以及它们如何互补。不论是开始者和结束者，还是纯化论者和实用主义者，他们会结合各自的不同技能促使项目成功。在考察其他个人或专业背景时，要有这种理解和认识。不同的看法可能各有好处，接受这一点非常有助于在不同群体之间建立同理心。

## 培养信任

信任和同理心紧密相关：一方面发展，另一方面也会发展。增加信任可以增强一个团队的弹性，如果没有信任，人们可能对他们的项目或责任领域过于保护，这往往会危害他们自己的健康或团队整体的生产力。

举例来说，假设一个系统管理员团队对他们的服务器过于保护，对任何可能接触这些服务器的人都不信任，所以限制只有他们自己这个团队才有权访问服务器。如果其他团队不能在这些服务器上安装必要的软件或部署代码，这个运维团队很可能成为瓶颈，这将是一个障碍，其他团队可能会对此不满，或者会想办法绕过这个障碍。这是一个很典型的例子，人们考虑严重割据的环境存在哪些弊端时都会想到这个例子。

如果是某一个人而不是一个团队过于封闭，这种负面影响可能更为明显。如果只有一个人了解或能访问某个系统，他们就会成为这个系统的单一故障点。倘若这个系统出了问题，而那个人生病了或者在休假，团队的其他人就会束手无措，在找到那个人之前毫无办法，这会导致团队的所有其他人都停滞在这里，或者那个人永远也不能（或

愿意)放假。如果有更多的信任,就可以在个人以及团队之间分享这些知识,分担相应的责任,这会增加组织整体的弹性。

培养信任有很多不同的策略,要建立一个真正协作的环境,信任是必不可少的。区别群体和团队的一个重要因素就是是否有信任。这里将要介绍的策略包括快速信任、自我表露、信任但确认和公平感。

## 快速信任

快速信任是短期或临时团体或组织中的一种信任形式,表示开始时认为存在这种信任,然后随时间加以确认。组织行为教授Dr. Debra Meyerson<sup>注6</sup>最早对这种行为进行了研究,在快速启动的小组或团队中,由于缺乏必要的时间来培养更长期关系中的信任(这里表示个人之间的任何关系,并不特别限定为浪漫关系),所以常常会使用这种快速信任。由于时间有限,团队成员开始时假设是可信任的,然后据其他人的行为确认和调整这种信任。

## 自我表露

有研究表明,有时自我表露是信任关系的一大特点。如果足够敞开心扉,能分享关于自己的信息,这会增加信任感,同时增加人们相互之间的亲密性,还能进一步加强合作和协作。当然,在工作中,自我表露要达到一种平衡。如果没有足够的表露,可能会滋生怀疑;人们会顾虑其他人是否有所隐藏以及他们是否可以信任。不过,太多或不合适的表露可能反而会破坏信任和可信度,这包括不适当的准许,或者泄露别人的秘密(可能是真的泄露,也可能被认为是这样)。

## 信任但确认

信任但确认常用来描述通常可靠的信息来源,但还需要额外的研究来确认其正确性,考虑分担专业职责时也可以使用这种信任但确认模式。有些人在允许别人访问他们完成的一个项目之前,会先等待后者“获得”信任,这些人会发现自己遇到一个鸡生蛋还是蛋生鸡的问题:由于人们还没有得到信任,他们就没有机会去获得信任,既然他们没有这样的机会,又如何得到信任呢?应当鼓励人们分担职责,首先给予信任,然后再进行确认。除了项目工作和职责之外,分享权力和决策能力时也是如此。

## 公平感

除了技术方面,在人员方面,信任对于组织的发展也很重要。公平感对于员工满意度

---

注6: Debra Meyerson, Karl E. Weick, and Roderick M. Kramer, *Swift Trust and Temporary Groups* (Thousand Oaks, CA: Sage Publications, 1996)。



非常重要，这意味着员工要能够确信他们得到公平的对待。在这方面，有一点会很有帮助，就是要建立正式的角色职位、职务级别和工资标准，另外在这些领域还要有充分的透明度。这会帮助人们了解他们的职位需求或加薪和升职的过程，相应地，这会大大减少可能影响信任的情绪，如感觉在升职中受到不公正的对待。

除了分担职责和分享资源，分担风险也很关键。如果两个团队分担一个项目的工作或者分享相应的资源，但是倘若出现问题时只有其中一个团队会受到负面影响，那么在不存在风险的团队看来，有风险的那个团队可能就是不可信任的。另外，这可能导致权力差别，没有（或有较少）风险和相关问题的团队会有更大的权力。

## 人力资源

关于协作，我们要讨论的最后一个问题是构建和维护软件的人员的成本，通常希望这些人全天候（24/7/365）在位。推动devops发展的一个决定因素就是当时的软件开发实践对软件或服务器运维人员有负面的影响。尽管持续集成和“基础设施作为代码”等方法对此有所改进，不过仍然存在另外一些问题。

## 可用性和维护

用户总是希望我们运行的Web网站一直可用，所以我们经常要面对这样一个问题：维护时可能需要关机或者会影响部分服务，那么什么时间完成维护呢？如果严格从用户的角度考虑，受影响的用户最少时完成维护比较合适，所以对于美国的一家公司，由于业务量最大的时段在美国的白天，所以可能希望在美国的深夜或清晨完成维护，但是如果一个公司的主要用户群体在亚洲，情况就不一样了。

不过，在完成维护的人看来，这可能并不理想。不只是因为人们不愿意早起或者熬到深夜，这里的关键问题是，如果人们得不到充分的休息，他们如何能保证警觉性、响应性和有效性。从运维人员的角度来看，应当在最警觉和清醒的时候完成重要的维护操作。如果根据用户的需求无法保证这一点，如一项维护任务花费时间太长，经济损失过大，仍有一些方法来降低维护人员的成本。

员工应当得到合理的加班补偿。如果这种加班会作为他们工作中常有的一部分，应当在招聘启示中清楚地指出，使人们能相应地评价这个职位是否确实适合他们。要鼓励人们关心自己和自己的健康。例如，完成深夜维护之后，要给他们放一天假休息。要尽可能确保足够充分地分摊维护任务，或者团队要足够大，使人们在轮班之间有足够的倒休时间恢复。取决于具体情况，还可以为之提供交通费或餐费。如果工作角色发生改变，有人需要在其职责之外加班工作，就要相应地调整他们的补偿。



## 工作与生活的平衡

规划来年的员工人数时，要记住一定要保证工作与生活的平衡，这很重要。与服务器的资本规划一样，人的资本规划同样非常重要。如果你的团队人员除了工作日正常工作外，还要经常晚上和周末加班才能达到预期，这意味着工作肯定很糟糕，人们的士气会很低落而且充满倦怠。devops就是要建立可持续的工作实践，如何保证人员实现工作和生活的平衡正是其中的一个关键部分。

尽管运维领域中的很多工作传统上都要求这种加班来提供24小时/7全天候覆盖，可能需要维护、待命或者非标准轮班，但要记住重要的一点，对于除了工作之外还承担很多其他责任的人来说，这种需求可能对他们存在无意的偏见。与有伴侣、孩子或肩负其他家庭责任的人相比，单身的年轻人（没有孩子，或者没有需要精心照顾的宠物）更容易把业余时间都投入工作。对于通勤时间比较长或者有健康问题的人来说，这种需求也会增加他们的负担。如果要培养和维护一个具有多样性和包容性的团队，就需要考虑到这些问题，还要考虑如何调整工作需求，从而更为包容。

## 团队规模的考虑

要有人负责对警报和事故快速做出响应，为此可能要轮班开机待命或者全天排班，这是要考虑的另一个问题。在这个领域，较大的公司往往比较轻松，因为比较大的公司通常会有多个团队分布在全球不同地方，所以实现全天候轮班相对容易。

对于这些较大的公司，分布在世界各地的多个团队（通常3个）分别在他们的正常工作时间工作，他们的地理位置相距足够远，一个班次结束时，下一个班次刚好开始。这些团队就能共同提供全时覆盖，而不需要人们晚上加班。

尽管中等规模的公司往往有一个由运维工程师或系统管理员组成的完整团队，他们可以轮班待命，但是较小的公司或成立不久的公司可能做不到这一点。也许你认为没有太多的运维工作，因此不需要一个完整的运维团队，但是不论怎样，都不能只让一个人连续待命。需要由多个人分担待命职责，让人们有机会恢复体力和补觉。



即使只是短期的，睡眠不足也会导致很难集中精力或者表现不好、易怒或者容易焦虑，而且高血压或心脏病犯病的风险更大。如果长时期睡眠不足，这些问题可能会综合爆发。

在了解组织整体的健康状况时，一定要考虑一般意义上个人的健康（特别是倦怠情

况)。如果太过看重公司的短期经济或物质利益，而忽视公司人员的长期健康，这会带来长期的损失。

## Sparkle公司的有效协作

快速审阅了关于MongoDB和MySQL的讨论记录后，General注意到并不是所有人都对MongoDB发表了支持或反对意见。另外，她没有得到足够的信息来确定到底支持哪一个策略。“Alice，我希望你与Geordie一起了解MongoDB的特性和优点，另外考虑继续使用MySQL有哪些好处。你们可以在本周演示会上做一些简单的演示，好吗？我们下周还会再讨论，然后再继续实现特定的解决方案。如果有人有其他补充意见，请给团队发email”，General说。

这天下午，Josie向团队发出了一封email。她仔细考虑了自己的工作时间和当前参与的项目，认为她有时间为这个项目提供支持：

我之前有使用MongoDB的经验。对于没有复杂事务的应用用例，MongoDB确实很容易上手，开发会很快。根据我们目前对这个项目的了解，我认为评论服务就符合这种情况。我想应当与项目经理保持同步，确保能全面了解项目的预期。还应当把运维团队纳入进来，使他们能提供关于这个软件管理开销的反馈。我可以根据需要提供有关这方面的经验。

与很多内向的人一样，Josie通常比较喜欢先整理自己的想法，再用一种书面的形式（如email）分享，不过她把this email抄送给了团队，使他们也能知道她的想法。基于Sparkle公司的“全部回复”邮件文化，Alice和Geordie都加入了讨论，另外Alice还把运维团队增加到电子邮件线索中。

## 小结

归根结底，一个devops项目要获得成功，很重要的一部分就是要让人们比从前更高效地一起工作。人们有很多不同的工作方式，特别是他们的工作风格和优先目标不同时，往往会采用不同的方式工作，不过要努力促使人们协作，这是让他们充分发挥才能的关键。要实现共同的目标，建立协作文化所需要的同理心，关键就在于沟通，对任何希望提高绩效的组织来说，这都是一个非常必要的技能。

单个成员之间的关系很重要，这些关系使社区相互关联并使人们可以从中受益。这些关系需要信任、同理心和互惠互利。如果成员之间有严格的等级划分，往往不太可能

建立有效的社区，而人们有不同层次的知识 and 经验时，这就意味着每个人都可以为整个公司做出某种贡献。

这在某些人看来有些费解，在他们的印象中，devops非常强调技术，不过应该记得，devops原先的目标就是让两个不同团队的人能够交流，devopsdays最初就是为了促成不同团队间的这种对话，并不断演进到今天。在第三部分中，我们将把这些概念从个人层次扩展到团队，直到最后扩展到组织层次。



# 协作：误区和问题排查

如果人们在专业和个人背景上存在差异，倘若不加处理会带来摩擦，想要有效地协作时，这就会产生问题。要帮助人们明确他们不同的动机和目标，并使他们积极为之努力，这对于有效地领导非常重要，作为领导者，可能有一个官方的领导身份，也可能只是在个人贡献者中有领导地位。

对于有这种领导角色的人，尤其是刚从工程部门转入管理层的经理（管理工作是一种职业改变，而不是一种提升），我们建议要用常规的办法来提升和保持管理技能。与其他技能一样，管理技能也需要学习、培训和实践来充分提高。一个好的领导能让身边的人展示出自己最好的一面，而不是只关注他们自己或者少数“超级明星”，这种领导才能往往是区分高效组织与低效组织的关键。

## 协作误区

人们在组织中会有自己的角色，为了更好地履行角色，或者在组织中更好地发挥作用，他们需要学习和成长，关于协作的很多误区就与人们是否愿意或者是否能够学习有关。

## 老系统管理员学不了新技术

关于协作能力，一个常见的误区是：一直固步自封的人（如BOFH，这是第3章提到的一个心怀不满的系统管理员形象）无法在开放或跨职能的环境中学习协作。另一方面，从业人员也有担心，如一些系统管理员尽管经验丰富，但更多地只是熟悉老式的

UNIX，他们担心必须成为开发人员才能保住他们的工作。不论哪一种情况，对于协作的devops环境需要哪些新技能，人们往往存在疑惑。

事实上，这些新技能绝对可以学习，不论这是新技术，还是“软”技能（例如，如何成为一个指导者，或者如何与组织中的其他人建立同理心），都是可以学习的。不过，要记住重要的一点，学习新技能需要花时间和精力，天下没有免费的午餐。如果你是一个管理者，希望培养你的团队成员，让他们不断成长，就要为他们提供必要的时间和资源（如提供图书预算，允许他们参加会议以及提供内部培训机会），而且不要期望一朝一夕就能做到。



每一个人学习新技能都需要时间和练习。不要认为年龄比你大、和你不同或者与你不同背景的人就没有能力学习。

如果你从另一个角度考虑，担心你的技能是不是很快就会过时，要记住，即使出现了新的基于云的技术以及无处不在的容器化，公司仍然需要各种不同的技能。关键是要找到一个合适的团队或位置。处在早期阶段的创业公司会努力让产品上线，在具备能力之前会把所有数据都交由Amazon EC2托管，这样的公司可能不会有一个专门的UNIX管理员或网络工程师职位，但是更大、更成熟的公司绝对会有自己的数据中心，而且不论有怎样的传言，大规模的公司和数据中心并不会很快消失。

## 想要快速成长就需要聘用超级明星

很多创业公司的创始人发展他们的新兴公司时，会特别强调写代码的能力。他们希望尽快完成产品构建，从而能获得用户或投资来保证公司继续发展。不过，绝对不能只考虑单纯的技术能力，因为如果你的公司在发展，就必须让你的团队也随之发展。有些组织认为在聘用人员方面不得不“将就”，因为他们可能没有足够的预算与更大更成熟的公司竞争，无法用高薪吸引人才，或者他们可能认为，为了让产品早日上线，需要聘用编程高手，即使这个人很差劲也是值得的。

这种想法的问题在于过于关注短期结果，公司早期的员工将确定公司的格调，而不论你是否希望如此。如果这些人太难共事，你可能无法聘用到你真正需要的人，或者最后你只能聘用和保留更难相处的人。尽管可以打破老习惯，可以学习新技能，这一点没错，但是我们常常看到，如果团队里有越来越多以自我为中心的工程师，人们不能很好地沟通，或者总是粗鲁地打断别人讲话，只是为了能更快地完成代码或产品，就容忍甚至鼓励这些行为，那么团队会变得越来越单一化。从长远来看，这是一种敌对



的文化，与之前介绍的信任、尊重和互惠的原则是对立的，这样的团队将无法很好地发展。

## 多样化的团队不能有效地协作

增加团队或公司的多样性时，人们常常有一个顾虑，担心这会带来更多的人际冲突。有些人指出会有“政治正确性”之类的问题，担心如果增加多样性，他们的员工就不能再随便开玩笑。对于前一个问题，尽管研究表明更多样化的团队在短期内可能有更多的冲突，但是这些影响会逐步消除，不仅因为从长期来看，多样化会带来更大的创造力和更强的问题解决能力，其利远远大于弊，另外团队成员会学习如何与更多不同的人协作和共事。

不仅如此，一个单一化环境中讲的笑话很可能涉及种族主义、性别歧视或者存在其他问题，不适合一般的工作场所。这些下流的玩笑可能会让一些人得到消遣，但有可能导致一种敌对、无礼的环境，以至于影响创造和创新，这就得不偿失了。

## 协作问题排查

处理个人之间的问题可能很棘手，最初正是为了缓解这些冲突才提出了devops的想法。

### 团队中有些人不尽责

与团队一同工作时，有一个常见的问题：可能会发现一些团队成员不尽责，没有完成他们的任务和承担的职责，甚至还可能带来问题，使其他人不得不腾出时间来帮助他们。如果团队里其他人提出这一点，或者如果当事人认识到没有达到周围其他人的水平，或者如果他们的经理注意到这一点，就说明存在这个问题。

要解决这个问题，第一步是明确角色和责任。如果一个人的角色很含糊，或者由于工作环境在不断演进，一个人的具体职责会随时间变化，就无法准确了解对他的预期。一旦明确了对他们的预期，接下来可以评估这些预期是否现实：他们是否具备必要的知识和资源来实现他们的目标？如果需要更多的培训或指导，就应当为他们提供相应的机会，而且要让他们有时间掌握所学的知识。如果未能提供完成所有工作所需的时间或资源，这个问题必须得到解决。对于相同层次相同角色的不同员工，可以比较他们的工作量，通常可以从中看出是否需要重新分配工作。



## 角色预期

就工作而言，角色是一种职能，包括一组特定的重复任务。根据角色所在的团队、组织和行业的文化，会对角色有特定的预期。例如，在一种文化中，经理、架构师、首席和项目经理都有相应的内涵，会影响人们对能力的评价。如果一个人是架构师，有些组织可能不希望他们考虑代码的问题，但并不是每一个组织都以这种方式定义架构师角色。加入一个新团队时，要了解这个文化中对角色的预期，这很重要。同一个团队和组织的角色预期也可能随时间变化。

如果有报告称一些人表现不佳，而且报告来自当事人以外的其他人，如团队伙伴或经理，就要做一些工作来明确这些说法是否准确。在这里，有时可能存在无意识的偏见，会影响我们对别人的看法，而我们自己并没有意识到。如果预期已经明确定义，是特定而且可度量的，这会很有好处。

除了这些环境方面的原因，可能还存在一些个人原因导致一些人未能达到预期的工作质量或数量，如：

- 个人或家人健康。
- 缺乏自主性。
- 缺乏奖励。
- 工作与加入团队时的期望不匹配。
- 目标变化很快。
- 工作不断改变。

一个好的管理者应当确保员工有适当的资源来应对这些挑战。可以采用以下解决办法：

- 远程工作。
- 灵活安排。
- 重新评估任务和劳动分工。
- 培训。
- 指导。

- 公开认可。
- 定期的一对一沟通，根据需要提供指导。

要当心出现倦怠，员工的身体和心理健康很重要。倦怠不仅是个人问题，这也说明了工作环境存在问题。

## 我们要决定是否辞退某个人

与表现不佳类似，另一个问题是要确定某个人什么时候不再胜任他的工作。我们相信人们能够改进，而且大多数情况下他们也都希望做出改进，但是也有例外，有时可能情况糟糕到已经无法解决。如果问题是不具备某种技能，而且至少已经尝试过一个培训或提高计划，要确保留出足够的时间来考察培训是否有效（如果少于一个月，这往往是不够的，取决于具体的培训，接近6个月可能比较合适）。如果培训只面向一种学习方式，进度会显著减慢。

建立一个发展规划时，如果能确保有足够的指导，这会增加成功的概率。一个好的指导者，特别是对其他人的职业发展提供过帮助的人，会很好地帮助我们扫清障碍。如果你知道组织缺乏资金或人力资源，不能切实地帮助培养和培训员工，就要让相关人员清楚地知道这一点，以便他们决定下一步怎么做。

决定辞退一个员工不一定是件坏事。在一个组织中，目标、价值观和优先事项要一致，这很重要，对于新生代的员工这一点尤其突出，他们往往有不同的个人目标和动机，所以会有不同的行为模式。另外，目标和优先事项可能经常随时间改变。可以利用与管理者或指导者的反馈面谈定期与员工交流，这会较早地发现这些变化和不一致，如果某个改变是合理的，就能更快地开始而不必拖延到以后。

不过，有些问题可能没有必要去尝试解决。如果有些人知道自己不喜欢在大公司工作，公司发展到一定规模时，最好让所有人都了解这一点，允许员工更早离开，并为他们提供有帮助的推荐信，而不是做无意义的尝试，试图解决这个无法解决的问题。来看另一个例子，如果一个创业公司被另一家公司收购，产品方向有很大改变，可以想见，肯定有些员工不同意这个新方向，在这种情况下，也没有必要强制他们接受。



### 文化契合和偏见

要记住最后一点，尽管前面都是不一致或不契合的例子，但是要仔细考虑“文化契合”的含义，避免有意和无意的偏见。文化契合并不表示完全相同，也不表示不允许有其他不同的观点。

## 我工作过度，压力过大，非常倦怠

如果你有倦怠的症状，很焦虑，很疲惫，而且不仅对工作的满意度下降，对你自己的满意度也在下降，就要尽早解决根本问题，这很重要。这种情况通常不能自行消除，你必须采取行动。

### 短期策略

从短期来讲，要尽可能想办法让自己得到支持，使你有机会“充电”：

- 放假，不去看工作邮件和聊工作上的事情。
- 把工作委托给别人或者干脆拒绝。
- 向专业人士寻求帮助，心理健康与身体健康同样重要。

### 长期策略

可以对你的所有责任做一个盘点，这也很有帮助：

- 对于每一个责任，试着分析是否可以采取行动来减轻这个责任带来的压力。
- 出现倦怠时，可能表现为一种一般性的压力或抑郁，所以要明确哪些特定方面造成了你的压力和焦虑，而你自己没有意识到。

### 找出责任点

导致额外压力的一个原因通常是感觉某个事情只是你一个人的责任。可能要由你一个人来完成某个项目，也许你的团队里只有你一个人，或者只有你一个人有兴趣改善组织的文化。不论是哪一种情况，都会增加你的压力和孤立感，这也是造成倦怠的一个原因。我们无法知道每一种特定的情况，但是有一点是明确的：没有哪个项目、工作或公司值得你牺牲你自己的健康。



任何时候都有可能出现压力和倦怠，不过在组织的拐点周围更经常出现。如果感觉工作过度，特别是如果同一个团队里多个人都有同样的感觉，这可能是一个信号，说明现有的人力已经无法完成所要求的工作，需要聘用更多人员。与你身边的人谈一谈，看看工作过度的情况是不是很普遍，相应地需要在更高层次上解决这个问题。

最后来看你的工作环境中哪些因素会带来压力。如果你发现自己经常抱怨某个工具，就有必要看看是否能对这个工具或有关的工作流加以改进，或者甚至可以把它替换为



其他工具。对于你使用的各种流程也同样可以这样处理。可以与团队成员沟通，看看他们有没有什么解决办法或者特别的技巧，另外要记住，这里还可以参考你的组织之外的方法，这会很有帮助。

## 团队里有些人感觉不被尊重

你可能在尽你所能地创建一个多样化而不是单一化的工作环境，尽管如此，仍会发现你的团队中有些人感觉不被尊重或者甚至感觉不安全。有些人可能感觉他们总是被人打断，或者没有人倾听他们的想法。例如，有研究表明，在工作场所中，女性比男性更常被打断，而且相比于男同事提出的建议，女性的建议也不那么受重视。

在建立一个多样化的团队时，这是很有可能出现的实际问题，应当提供必要的资源来支持这一目标。要鼓励（甚至要求）员工（尤其是管理者）针对普遍的多样化问题进行培训和教育，这包括敏感性、联盟能力和行为以及无意识的偏见。要以身作则，从上向下进行。另外要对少数群体中的成员尽可能多地提供支持，其形式可以是公司提供资助，或者建立社交群体，也可能只是确保员工知道他们的法律顾问和HR联系人是谁。创业公司应当尽快成立一个HR部门。

如果确实出现了问题，就需要进行协商，具体的协商技术超出了本书的范畴，这里不做详细介绍。不过，总的来说，要用心提高你的倾听能力，甚至（尤其是）当你听到的内容让你感觉不舒服或者在你的意料之外，也要能耐心地继续倾听。没有人愿意听到自己的同事或朋友在骚扰别人或者有其他不良行为，不过由于勇于发声会承担风险，所以诬告的比例会很小，更多的是一些骚扰或霸凌的行为未能报告。

要注意，如何解决这些问题会对你的组织文化有深远的影响。如果有人在骚扰其他人，倘若还允许他继续工作，其不良行为没有对他带来任何影响，这就会向其他成员发出一个明确的讯息，让他们感觉在这个公司里不安全。如果遭到无礼对待或被骚扰的报告被驳回，没有得到充分的重视，人们就不会有安全感，这会制造一种不好的氛围，等将来发现问题时可能就已经太迟了。



防范有害的文化相对比较容易，而一旦信任被破坏，修复起来就要困难得多，正如我们在这一章和前一章中讨论的，要建立一个有同理心和协作性的环境，信任非常关键。

## 有些人沟通得不够

想要打破或防止出现孤岛时，常常会抱怨人们沟通不够，可能是团队内部沟通很少，

也可能是团队之间没有充分沟通。假设已经明确了期望并已经做了解释，这种情况下最好的做法是分析哪些因素可能影响人们的沟通意愿，见前面“如何沟通”一节中的讨论。

如果个人不愿意沟通，可能存在信任问题。这种情况常见于问责文化，在这种文化中，人们总是负面地看待失败，而且常常因为过失而解雇或处罚有关人员。一段时间后，人们可能越来越不愿意交流（这是可以理解的），而且可能会犹豫是否要提供额外的信息。要转变这种文化需要一定的时间，人们必须看到勇于发声会得到褒奖而不是处罚。可以以身作则，在组织中开展公开、无问责的事后分析，这会很有帮助。

可能还存在另外一些问题，有些人的回应方式会让其他人感觉很无礼，这可能是有意的，也可能是无意为之。email和其他纯文字沟通通常会给人一种唐突或傲慢的感觉，因为文字沟通很难传达语音语调。或者，可能某个团队成员对别人表现得很无礼，让别人很讨厌甚至完全敌对，不过他可能并不是针对所有人，或者不是在所有情况下都这样，所以这种问题不是每一个人都能明显地看出。如果有些人不能态度很好地与人沟通，最终将与你的组织文化不相适应。

最后，各种文化差异和个人差异也会有影响。在跨国团队中，由于有不同的文化习俗，有些人看起来可能更沉默寡言或者说话很简短，而对他们来说，这就是最合适或最礼貌的沟通方式。不同的工作方式或个性也会带来影响，有些人可能比较保守，不愿意太多地谈到他们的工作或成就，担心在别人听来像是在自吹自擂，而另外一些人可能更外向，更喜欢与别人交流。要让人们有机会解释他们的偏好，人们有不同的偏好时，允许团队讨论可以采用哪些沟通媒介，这会很有好处。

## 一个员工（或应聘者）技术很强，但个性很古怪

每个人都不希望做出错误的聘用决定，正是因为这个原因，我们会在筛选和面试过程投入很大的精力。不过，有时人们会不遗余力地证明为什么有必要聘用某个人，尽管这可能是一个糟糕的聘用决定，一个常见的例子就是聘用一个“技术超群但社交能力差的”工程师。之所以要聘用这样的人，有些人给出的理由是他们在技术上的贡献极大，远远超出他们在人际交往方面不良行为带来的负面影响。一个经典的例子就是Linus Torvalds，他讲话很生硬，但人们能容忍他，因为是他创造了Linux。

我们在前面提到过，你没有必要成为所有同事的好朋友，但是确实要考虑负面行为可能产生的影响。这种影响不一定容易度量，不过在做这种聘用决定时，可以先问这样几个问题：



- 多少人会因为无法忍受与这个人共事而辞职（或者一开始就不会到你的公司应聘）？
- 与这个人的争执或其行为的其他负面影响会让多少人的生产力受影响？有些人可能认为“脸皮厚一点”就行了，但要记住，这并不是一个可接受的反应。任何人都不能忍受在一种恶语相加的环境中工作，也许你个人没有亲眼看到这种情况，但并不表示它没有发生过。
- 如果这个人的行为会影响女性、有色人群或者少数群体的其他成员，这对你的组织的包容性和多样性会有多大的伤害？要知道，还存在反向的沟通渠道，如果管理层对这个人的行为不加干涉，很快就会口口相传，尽人皆知。
- 由于这个人拒绝写文档，你要浪费多少时间调试他的代码？或者由于他们不遵循组织的测试流程，你要多花多少时间来处理他们推送的提交？

很多情况下，这种人会对组织带来伤害，包括技术上和文化上的危害，这并不“值得”，特别是你的组织会在业界有一个坏名声，人们会认为你的组织容忍（甚至鼓励）这些不好的行为。一般来讲，做一个聘用决定时，如果人们说“嗯，他就是那种人，不过……”，你完全可以不听“不过”后面的话，这个人不合适，你应当寻找那些不会背离你的组织文化的应聘者。

## 我感觉在目前的团队/组织里没有发展

我们考虑自己的情况时，所想到的最重要的就是我们的职业发展和职业规划。有时，你可能感觉在当前位置上止步不前，无法进步。有些情况下，可能需要人们注意到你的成就，也许你已经满足晋升所需要的全部条件，但你不希望太浮夸，所以除了你以外，没有人注意你做了哪些工作，管理者也不知道你所做的工作。

另外，有些情况下可能存在有意和无意的偏见。例如，一个组织的晋升流程中，可能把“其他X级工程师同意这个人得到晋升”作为一个重要的条件，但是如果X级工程师小组的人员组成很单一（例如都是男性），他们就可能无意间倾向于晋升与他们类似的人。如果能与HR一起确定不同人群的晋升比例是否有差别（结合HR的经验），将有助于在组织层次上解决这个问题，当然，这是长期收益而不只是短期好处。

要记住指导和赞助之间的差别。关于这个主题的讨论已经有很多，Ride的移动工程主管Cate Huston做了一个很好的总结，她说赞助与拥有权力有关，有权力的人通过赞助来支持权力更小的人。指导者会给你提供建议和指导，赞助者则是全力支持你的人，如果能找到一个好的赞助者提供支持，会帮助你得到一些迫切需要的引导。



不过，如果这些你都已经考虑过，仍然认为“组织/经理偏心，对我不公”，而且你并不是边缘化群体的一员，就需要考虑是否还有一些需求你还不满足（可能是明确的需求，也可能不那么明确）。这不是说男性（作为主导群体中的一员）不会受到不公正的对待，不过确实有可能是你自己的行为存在问题，要找出这些问题并予以解决。批评其他人或者系统可能很容易，但如果要承认是你自己错了或者在某些方面有欠缺，就没有那么爽快了，不过作为成熟的工程师，自省和诚实的自我评价是必须具备的一个特点。



可以退一步考虑你的行为对其他人有什么影响（这包括有意和无意的行为），以及这种行为与你的组织的价值观是否一致。如果你的组织认为同理心是一个核心能力，或者有一个“请你止步”法则，倘若大家都知道与你共事很麻烦，这会阻碍你的发展道路，就像你在技术上有缺陷一样，甚至可能更糟糕。

很多情况下，更高级的职位不仅强调技术能力，还会强调人际能力，需要乐于助人，能够赞助和指导别人。与其他人和睦相处，而且能很好地处理冲突，每个人都应当注意这些方面。

## 没有人听我说

在小公司里，如果你有问题，可以径直走进CEO的办公室与他直接交流，而较大的组织与小的创业公司不同，在大公司里，你就像一个庞大机器上的一个螺丝钉，没有人能听到你的声音。如果你的公司正处在发展阶段，也会有这种情况，可能你的部门发展得非常庞大，人们的影响力远不如前，也可能你有了一个新经理，或者合并或收购也可能对文化产生显著的影响。



根据你的直接经理提供多大程度的支持，可以选择不同的做法。如果你的经理能提供相当大的支持，可以告诉他你希望对团队和组织有更大的影响，问他有什么建议。特别是如果你是一个初级员工，或者刚刚加入这个团队或组织，你会发现改变工作或沟通方式可能很有帮助，另外在组织里找一个指导者也能帮助你培养这些技能。

如果某个人或某个团队根本不听你说什么，这说明可能存在一些人际冲突需要处理。可能是有些人不好相处，不过也有可能是因为之前遗留的一些问题，需要先处理好过去的某个冲突。我们都是普通人，有时会有情绪，这会影响我们与其他人的相处。如果你的某个请求无意间冒犯了一个人，或者你承诺交付一个产品但是没有履行你的承

诺，就会留下不满，需要消除这种不满。承认自己犯错可能不容易，不过要注意，你的行为对你的工作关系有很大的影响。

如果你的直接经理不听取你的意见或者不尊重你，可以尝试先直接与他交流，因为他们可能没有注意到这一点。可能还需要让他们的经理或一个HR代表介入，来帮助处理这个问题。新任的经理要面对越来越多的下属，而他可能没有这方面的经验，所以管理培训或团队重组对此可能会有帮助。不过，如果所有这些都没有用，你的经理或团队仍然不理睬你的意见，说明你所在的环境是一种不健康的环境，最好的办法可能是离开这个环境，寻找一个新职位，可能在当前组织中另找一个职位，也可以寻找其他组织中的职位。

## 我们刚刚重组或裁员

在一个组织的生命周期中，裁员是一个很自然的阶段，这个阶段往往会减少生产线或员工人数。取决于你的环境文化，这可能让人很有压力，特别是如果你之前没有经历过裁员，可能会很担心。

可以问自己几个问题：

你知道为什么会重组或裁员吗？

如果领导清楚地说明了发生这个变化的原因，你不用自己猜想遗漏了哪些信息。

如果你不知道原因，可以询问你的领导。注意，这并不表示你必须接受或同意他们所说的原因。有时领导也会犯错误。失败是一个学习机会，要注意持续的失败，这可能说明存在更大的问题。

如何通知重组或裁员？是否及时？

领导要及时地通知每一个人，而不是让消息先泄漏到媒体，或者几个月之后人们才得到消息，如果能及时掌握情况，你就有时间妥善安排，而不会等到人们纷纷询问时还毫无头绪。这里体现了领导对这个过程的关心。

你的工作会有很大变化吗？是否要向同一个经理报告？岗位职责、工作难度或者决定权会有变化吗？

如果你的工作实际上是一样的，而你只是换个名头，这就没有什么变化。花些时间好好了解为什么会发生重组或裁员，哪些人会受影响。这些会给出信号，从中可以看到公司的关注点是什么。如果你的工作会发生很大改变，则要从一个应聘者的角度研究你的职位（加入一个公司之前，应聘者肯定会仔细评估是否接受录用）。你的价值是否被低估？这个改变是否会影响你的学习和经历？你与新经理是否能很好地相处？



你在过去12个月里是否经历过多次重组？公司的生命周期中定期有一个裁员阶段。

如果这是一段时间以来的第一次重组，这可能只是领导层变化的一个自然演进。

这是健康的，可以在组织中纳入新观点。通过减少脆弱性，这会尽可能降低组织的风险。如果你经历了多次重组，就需要更深入地挖掘其原因。之前的重组或裁员确实完成了吗？或者这是否只是一个漫长而迟迟不能终结的过程？领导可能会犯错误，有时需要一段时间重新洗牌。分析这对你的健康和工作价值有什么影响。衡量重组提供的机会是否大于所带来的人事成本？

你的团队是否人力不足？

如果你的团队在重组或裁员之后仍然有足够的人力，能正常完成工作，工作压力没有变化，那么重组同样不会对你有影响。花些时间来了解组织中哪些人会受到影响，并了解这说明了什么。如果你的团队人力不足，你要马不停蹄地四处救火，这就会影响进一步提高你的技能，甚至还会退步。从内部来看，这样救火似乎很好，但是从外部看，这只是不求进步的借口。



根据对这些问题的回答，你可以确定重组或裁员与你对组织文化的理解是否一致。每一天结束时，你要仔细选择在哪里工作，并考虑为什么这样选择。不要只是因为有大笔分红就留在某个公司，在这个行业里这被称为“黄金手铐”。如果为了钱留在某个公司，而以你的健康为代价（因为压力在不断增加），你会发现这根本不值得，你绝对不会再做这样的选择。

要记住，你完全可以选择辞职。不是每一个团队或组织都适合每一个人。并不是所有人都关心devops，或者能够在其他领域发挥你的作用，对所有人来说，有时最好的事情莫过于找到一个真正适合的位置。



# 亲密性：从个人亲密性

从即将公示到鲜有者，很多人都只强调团队内部的关系，不过，团队、组织部门甚至公司之间也存在关系，这些关系会影响工作的速度和价格。1973年美国社会学家Mark Granovetter在他的文章“弱联系的力量”<sup>[1]</sup>中阐述了这些关系的重要性，个人之间弱联系的机会，以及在这些不同联系上信息如何流动。

## Sparkle公司开发演示会

由于Jessica没有使用MongoDB的经验，她能帮助Alice和Geordie很快搭建两个虚拟机，用来比较和演示MongoDB和MySQL。他们创建了一个演示应用，这是一个简单的基于Web的照片应用，提供了一个评论特性。

项目经理Hedwig和运维工程师George参加了开发团队的演示会，希望了解这个团队的过程，在更真实的情形下查看这两个产品的系列。

“我真的很高兴能这么快用MongoDB开发这个应用。关于这个评论平台，我们实现的很多特性这个JavaScript框架中都已经有了，它与MongoDB可以无缝地协作。这就为我们留出了开发时间，可以专心考虑如何避免恶意和垃圾评论，感谢Alice很给力。”

“我发现，已经有好多工具可以用来做这个应用，甚至不用写多少代码。我们使用MySQL，Alice帮助我了解了快速搭建的数据库，感谢Geordie和Hedwig很给力。”

[1] Mark Granovetter, “Economic Action and Social Structure: The Problem of Embeddedness,”

# 亲密性：从个人到团队

从招聘启示到绩效审查，很多人都只强调团队内部的成功。不过，团队、组织部门甚至公司之间也存在关系，这些关系会影响工作的速度和价值。1973年美国社会学家 Mark Granovetter 在他的文章“弱联系的力量”<sup>注1</sup>中描述了这些关系的重要性、个人之间强弱联系的组合，以及在这些不同联系上信息如何流动。

## Sparkle公司开发演示会

由于Josie过去有使用MongoDB的经验，她能帮助Alice和Geordie很快搭建两个虚拟机，用来比较和演示MongoDB和MySQL。他们创建了一个演示应用，这是一个简单的基于Web的照片应用，提供了一个评星特性。

项目经理Hedwig和运维工程师George参加了开发团队的周演示会，希望了解这个团队的进展，在更真实的情形下查看这两个产品的差别。

“我真的很高兴能这么快地用MongoDB开发这个应用。关于这个评论平台，我们谈到的很多特性这个JavaScript框架中都已经有了，它与MongoDB可以无缝地协作。这就为我们留出了开发时间，可以专心考虑如何避免攻击和检测评论里的霸凌行为”，Alice很激动。

“我发现，已经有很多工具可以用来设计和规划所需要的拓扑结构，还能升级和监控MySQL。Alice帮助我了解了协调运维团队使用MongoDB有哪些好处”，Geordie指出。

注1： Mark Granovetter, “The Strength of Weak Ties.” American Journal of Sociology (May 6, 1973)。

此时，Sparkle公司开发团队可能决定继续使用MySQL。他们会向运维团队调查，更多地了解引入新软件可能带来的成本。他们也可能把决定权交给项目经理Hedwig，他了解客户有哪些期望。下面来深入分析建立亲密性将如何增强我们的组织并帮助我们做出重要的决定。

## 人际关系网

Granovetter描述了人与人之间3种不同的联系：强联系、弱联系和无联系，根据他的描述，联系的强度是时间、情感强度、亲密性和各个联系互惠性的组合。他指出，尽管强联系和弱联系共同将整个社会关联在一起，但是在分享资源和信息方面，弱联系比原先所想的作用更大。例如，他调查了一些找工作的人，超过一半的被调查者都是通过他们用弱联系而不是强联系认识的某个人找到了工作，这个人只能算是一个熟人而不是关系很深的朋友，可能两三周才见一次。

在我们的工作生活中，我们与我们的直接团队就存在着强联系，每天我们与他们一同工作，在同一个办公环境中，和他们一起喝咖啡。在这个紧密群体之外的人往往就是弱联系或者无联系。正如Granovetter以及之后其他研究人员和社会学家所指出的，通过培养我们已有的弱联系，可以从中获得很多好处，这可能是本公司里其他团队的人（但交往不多），也可能是其他公司同一领域的人。这一章会介绍团队如何运行和交互，以及我们如何使用这些联系建立更强的工作关系。

## 怎样才算是团队？

为什么有些人看起来总是与团队或组织目标格格不入？要了解这一点，我们不仅要理解团队的组成，还要理解个人与所在团队的关系和对团队的认同感。

团队是朝着一个共同目标努力的群体，人们相互之间彼此依赖，成员之间有一定的熟悉程度。团队会有自己的一组信念，反映了大多数成员的想法。另外，有内部和外部作用力促使团队建立和维护其团队同一性。

## 团队要做的工作

团队的工作通常分为5大类<sup>注2</sup>：

注2： Scott Belsky, “The 5 Types of Work that Fill Your Day,” <http://bit.ly/5-types-of-work>.



### 反应性工作

这些工作是对其他事件的响应（如响应电子邮件或收到的客户ticket），而不是主动完成的工作。

### 计划性工作

在安排和确定其他工作优先级方面所做的工作。

### 过程性工作

维护工作，如跟踪之前发出的电子邮件或填写开支报告。

### 不安全性工作

由于我们自己的不安全感而做的工作，如在线检查我们的个人或组织声誉。

### 问题解决性工作

需要创造力和注意力的工作。

## 定义亲密性

还有第6类工作，称为关系性工作。关系性工作是一个社交催化剂，可以促进工作，从而缩短完成其他工作的时间，减少沟通壁垒，并基于声誉建立信任。亲密性是对个人、团队、企业部门甚至公司之间关系强度的度量。准确地度量亲密性极其困难。

不论组织有怎样的结构，公司往往会为各个角色指定某种价值，建立一个声望阶梯。有些人可能认为某个角色比另一个角色更有价值，这就创建了一个重要性层级结构。公司文化会根据谁拿到资金、得到晋升或获得其他想要的结果强化这个层级结构。基于变化的角色，这个阶梯可能会产生一种错误的向上流动性的感觉，鼓励个人追求那些他们可能并不适合或者不太喜欢的工作。

公司通常有多个需要一起工作的团队。另外公司开始依赖外部服务时，团队还需要与其他公司合作来达成目标。要了解谁拥有所需的知识或技能，并与这些不同的团队、组织和公司建立强关系，这就是亲密性。这一章将分析可能促进或破坏团队间积极合作的人为因素，讨论有效合作的策略，并介绍这些因素如何在组织层次上帮助维护devops组合。

## 团队内的人际联系

内力是稳定性和变化之间的一种对抗。一个群体中的个人可能倾向于稳定性，着力寻求不可变的真理或“最佳实践”。由于我们倾向于稳定性，这会让一个团队或群体更

有内聚力，另外由于人们很自然地避免冲突，这会让一个团队中的人比不在同一个团队中的人更容易相互认可。当然，这有好处也有坏处。如果一个团队有太多的冲突，可能没有足够的内部稳定性来实现它的目标，但是如果过多地避免冲突，又会导致一种思维同一性，这会降低创造力和问题解决能力。

将团队组织到一起的外力往往在具体环境中形成。如果与另一个群体发生冲突，如竞争某些共享资源，这可能会把团队紧密地联系在一起，并加强团队认同感，因为个人的成功和认同感与团队的成功和认同感通常是一致的。外部给出的目标或最后期限可能也有类似的效果，同样的，如果有些问题威胁到团队的稳定性甚至存在性，也会有这种作用，不过需要指出，尽管有些外部情况和冲突会促使团队凝聚在一起，但是如果一个群体缺乏足够的初始内聚力和稳定性，或者已经存在大量冲突，出现新的外部情况和冲突时可能就无法按原先的方式继续存在了。

前面已经讨论过，人际联系的强度可以用多个因素来度量：

#### 共处的时间

只要一起工作一段时间就能帮助增强工作关系。

#### 关系的强度

克服挑战和解决冲突可以把人们凝聚起来，尤其是一些紧张的经历，如解决一个生产中断问题。

#### 分享故事的互惠性

公开和分享个人的成功与失败经历是相互认识的一种很好的方法。

#### 支持的互惠性

假期值班、同伴休假时换班，或者帮助某个人解决一个棘手问题，这些都是相互支持的好办法。

所有这4个因素结合在一起才能真正建立一个强联系。如果只是与某个人共处很长时间，这还不足以真正培养一种强关系。如果一个队友总是开玩笑或者说一些挖苦的话，而从不打开心扉或者讲真话，我们通常不会愿意认识或接近这样的人，另外如果一个队友不喜欢提供支持，如不愿意换班或提供代码审查，我们遇到难题时也不会去找这样的人。

这并不是说人们必须非常亲密或者在工作内外都要与队友成为最好的朋友。分享故事不是要谈论一个人的家庭或亲密关系，而是要与人分享在你的职业生涯中之前做过的某些努力、你不得不做的某个艰难的职业选择或者你一直渴望参与的项目。

## 团队文化

团队的文化会在很大程度上影响团队工作的有效性和团队成员之间联系的强度。不是从“文化契合”的角度讨论文化，也不是看你的队友是不是你想要推心置腹的人，更重要的应当是强调文化作为一个团队共同的价值观，以及这些价值观在实际中如何体现。

如果能清晰地传达共同的价值观，这会是团队成员之间一个强有力的维系力量。确保大家都理解团队的价值观，这对于避免或解决人际冲突很有帮助。例如，如果两个成员共同参与一个项目，而且已经临近这个项目的最后期限，他们之间就可能产生冲突。一个人可能认为他们应当继续，尽其所能地提交产品，尽管可能还没有充分测试或者特性还不完备，因为他们更看重履行承诺和满足最后期限的要求，而另一个人可能想先等等，他更强调交付满足质量标准的工作成果。在这种情况下，谁是“对的”很大程度上取决于在团队整体来看哪些价值更重要。

团队的价值观不能与整个组织或公司的价值观相冲突，也不能与团队成员的个人价值观冲突，这很重要。在这个领域中经常会出现“文化契合”的概念。我们将在第五部分中讨论，这个概念也有自己的问题：它可能过多地用来描述一些常见的活动，如喝啤酒或者都是某个球队的球迷，而不是表述共同的目标和价值观。



评价某个人的价值观与团队的价值观是否一致时，一定要考虑“价值观”和“契合”是否得到准确的使用，或者是否只是使团队变得更单一化。

### devops反模式：文化契合

General在一家创业公司工作，这家公司正积极聘请更多的工程师。这个公司是几个朋友联手创办的，同时他们还是老同学，多年前他们上了同一所大学，所以办公室气氛总是很友好。员工下班后常常一起去喝啤酒，特别是那些老同学们，据说他们甚至还会在办公室一起喝啤酒。

应聘者参加面试时，如果他们在早上的技术面试中表现不错，团队会邀请他们共进午餐，看他们是否能做到文化契合。由于办公室“酒友”群体的不断扩大，文化契合变成这样一个问题：“我喜欢和这个人一起喝酒吗？”General发现这个团队拒绝了几个条件相当好的应聘者，只是因为他们午餐时没有要啤酒。由于这个原因，大家认为他们可能不适合这个团队。



这个故事最明显的启示是，如果只有当人们愿意在工作时喝啤酒才适合你的团队，这说明你的团队可能很有问题，不仅如此，这里还有很重要的一点，也就是文化到底是指什么。不要考虑文化，你应当考虑价值观。对你的组织和团队来说，哪些价值很重要？应当考虑通过哪些方式体现这些价值，而不是看应聘者是否要喝啤酒。

价值观可以采用多种不同的方式传达，不过归根结底，人们要了解团队的价值观，主要途径还是考察团队整体的行为。事实胜于雄辩，你做的比你说的更重要。如果你声称非常看重可靠性，但是经常延迟发布软件，或者软件总是有很多bug，或者这二者兼而有之，人们会更强调这些结果，而不是你嘴上讲的价值观。或者如果一个组织声称很重视多样性和包容性，但是在正式的公司活动中请来脱衣舞女跳舞，人们肯定会质疑他们的价值观到底是什么。

这种价值观的传达不仅发生在团队内部，也发生在团队之外。在一个环境中，如果你的团队完成的工作或者他们提供的服务要以某种方式与其他团队的工作交互，能够在团队之外传达你的价值观就非常重要。要确保其他团队了解你的价值观是什么，你设定的标准是什么，以及你对工作和行为做出了哪些承诺。

可以察看其他团队的人选择以何种方式与你的团队成员交互（或者不交互），从中就能很清楚地看出这一点。如果人们都避免与你的团队成员交流，因为他们有不好的名声，被认为过于苛刻而且很难共事，或者人们会绕开或违反你的团队规则，因为这些规则会阻碍其他人的工作，你要向组织中的其他团队展示哪些价值观呢？

在团队内部传达价值观可以帮助团队有更统一的认识和行动，特别是有新的团队成员加入时，如果缺乏一致性，不论是不同团队成员展示了不同价值观（可能是与他们共事的人，也可能是他们学习的对象，不同的人可能传达了不同的价值观），还是团队所声称的价值观与实际行为不匹配，都会让新成员更难进入状况，不知道该如何工作。

除了缺乏一致性，缺乏责任感也会严重破坏其他人对你的团队的信任，还会破坏团队成员之间的信任。团队内部和外部的人注意到某个问题或违反价值观的行为时，他们会如何沟通？这并不是鼓励一种问责环境。实际上，出现问题时，问责是要找出某个人为此付出代价，而责任感涉及所有权、坚持和学习。

最近一项研究表明，如果团队成员相互之间以及与他们的领导有共同的价值观，与没

有做到这一点的团队相比，这些团队会有更高的内部信任度。<sup>注3</sup>信任可以带来更高的团队和个人绩效，这包括团队成员之间的信任以及对团队领导的信任。这种信任带来的强联系是一个团队或组织最大的竞争优势之一。要维护一种无问责、强调学习的环境，团队的价值观和文化必须匹配。

## 团队凝聚力

团队凝聚力是各个成员希望为团队做出多大程度的贡献，使之能作为一个功能单元继续工作。凝聚力差的团队会有一种“人人为自己”的心态，团队成员会更多地考虑个人利益，而不是从团队整体利益的角度来考虑。毋庸置疑，在凝聚力差的团队中，信任更少，知识分享也更少，而且不太会有同理心。

团队动力还有其他方面，包括团队整合（team integration），即一个团队如何将个人的经验和工作结合到统一的整体中，另外还包括团队的团结（solidarity），或一个团队是否有可以将成员凝聚在一起的共同利益。团队整合对于生产力尤其有意义，因为通过团队整合，个人之间可以分享更多的知识，使人们更多地合作，而不是把知识据为己有，这种思想聚合可以使一个团队更强大，更有生产力，而不只是单个成员的简单累加。

团结会让一个团队动力十足，感觉是一个最强阵容，不过有一点很重要，这种感觉不能矫枉过正。

有些说法会对其他团队做出价值判断或者表达某些成见，从中可以观察到个人中心主义观点。这样的例子有很多，如一些公司的“开发”和“运维”团队之间存在冲突，在这些公司里，常常会把问题扔到“隐喻墙”对面，称这是其他人的责任，或者会经常抨击其他团队完成的工作。在组织中的任何团队之间都能观察到这种情况，特别是采用某种角色层级结构的组织。

## 内群体/外群体理论

外部压力可以增强团队内部的凝聚力，这种想法几十年前就已经有了。通常这称为内群体/外群体理论，有人对此做了最好的解释：“团队里的同事友谊与来自其他团队的压力是相互关联的。外部压力的紧迫性会促进内部团结”。

著名的Simmel规则指出，“一个团队的内部凝聚力视其外部压力的强度而定”。在较小的团队中，这种凝聚力更强，正如他指出的，这是因为，较小的团队中，个人相互

---

注3： Nicole Gillespie and Leon Mann, “Transformational Leadership and Shared Values: The Building Blocks of Trust,” *Journal of Managerial Psychology* 19, no. 6 (2004)。

之间可能更相似。他相信，团队间的冲突会强化和维护团队疆界，将原本可能相互没有关系的人聚集在一起。

这个规则的一个推论是，根据这些外部压力或冲突的来源，可能会出现不同的团队。例如，假如管理层制定了不切实际的最后期限和预期，开发人员遇到问题时，可能把运维工程师看作是这些问题的根源。在这些情况下，内群体和外群体的对立往往很紧张，实际上也就形成了特定的社会环境。

如果应用到工作环境中，可以看到，组织中不同群体或团队之间出现的冲突可能带来利益冲突，不同团队会不断相互竞争。我们希望其他群体来帮助我们自己群体（或团队）中的成员，而且团队成员身份可能成为我们工作认同感中非常重要的部分。人们的态度往往是指责其他团队人员，系统管理员有时就是这样的，他们把所有非技术计算机用户都叫作“一般使用者”。很自然地，这种想法无法使他们成为一个有凝聚力的组织。



要解决群际冲突（或者至少尽可能减少影响），一种方法就是分享经验。如果能跨群分享经验，可以减少将来不同群体之间出现冲突的风险。前面已经提到过，分享经验是建立信任的关键，而与某个人紧密合作（即使只是暂时地，甚至在一种看似无关紧要的情况下合作），都会减少偏见，避免正常情况下很可能出现的问题。以类似devopsdays大会等场合为例，分享故事就是这些大会很重要的一部分，这可以让人们相互之间更好地培养同理心，从而更有效地共同合作。

如果为员工提供机会并鼓励他们参加多个不同的工作群体，这对于尽可能减少外群体的偏见也有很大帮助。这称为交叉分类，由于人们同时属于多个不同的群体，内群体/外群体的差别就没有那么重要了；要让人们知道与他们共事的人是多维度、多方面的，同时增加更多不同类型人群之间的人际交互、沟通和信任。有一点很重要，这种交互不是强制的，而是允许自然地形成一些利益群体（可能是一群一起远足的同事，或者是某个特定编程语言的粉丝群体），这就会产生组织亲密性。

## 检查群体从属关系

创建软件的人远远少于使用软件的人，这些创建者的群体从属关系和社会联系会对使用者产生深远的影响。例如，很多社交网络已经开始强制“实名政策”，会采取一些措施，如暂停没有使用合法名字的用户账户。这种政策有很多问题，其中最大的问题之一是：这会影响到甚至伤害未使用身份证上真名的人，如某些行为的受害者，他们可能想努力躲避某些类型的网友，他们可能还没有在生活中全方位地公开自己的身份，这些人也可能不愿意使用真名。



不同平台上这些政策的应用可能并不一致，一些名人通常会被区别对待（像U2乐团的波诺或麦当娜等知名艺术家就是很典型的例子），这会对已经身处危险或者在某种程度上被社会边缘化的人群带来最负面的影响。不仅如此，实名政策对于杜绝垃圾消息、减少虚假个人信息或者防止信息滥用并没有太大用处，尽管人们常常以此作为执行这些政策的理由。

之所以决定采取这种政策，很可能是因为主要决策群体中没有在网上不使用真名的人，这些人才会真正理解人们不愿意在网上使用真名肯定有某种正当的理由。让每一个少数群体都有成员参与每一个软件项目是不实际的，不过可以看到，决策群体的多样性越低，对于这个群体之外的人和用户来说，这些决策就越有可能产生严重的负面影响。

有一点很重要，我们要经常问自己：想要解决哪种问题？我们确实在解决正确的问题吗？我们的团队有没有必要的知识和经验来确认这一点，能否了解解决方案可能带来的反应？如果一个团队要处理社交网络上的问题，而团队中只有男性，他们对这一类问题没有任何经验，对其他群体中成员遇到的问题完全没有概念，根据他们有限的经验和理解，他们真的能提出最佳的解决方案吗？

我们要努力打破群际壁垒，并增进团队间的沟通和分享经验，在这个过程中，我们与人们的弱联系数也在增加，正常情况下这些人原本在我们的圈子之外，如果没有这样的途径，我们与他们之间可能没有联系。我们可以从这些弱联系获得知识和经验，这些会让我们创建更强大、更有创造力的解决方案，而且开发的产品不会无意间伤害部分用户。

## 扩展群体从属关系

原先的devops概念确实只考虑了开发人员和运维工程师。这是有道理的，因为devops运动就是这些领域的相关人员发起的，他们也最能感受到这里的艰辛，另外有一点是肯定的，devops强调的摩擦和割据问题确实是需要解决的实际问题。

不过，如果止步于此就过于简化了。大多数软件不是为了编写软件时的快乐而写的，尤其是那些要由运维团队部署和监控的软件。大多数情况下，公司会以某种方式销售这个软件（可能公开发售，也可能通过其他途径），而且公司是否得以生存发展就取决于这个软件。但是，倘若忽视公司的其他方面，将对所有相关人员都不利。

我们希望进一步扩展这个概念，而不仅限于开发和运维团队。信息孤岛、问责文化、无效的沟通以及缺少信任都是组织中很严重的文化问题，这些问题可能出现在开发和运维团队之间，也可能出现在公司的其他领域。作为成员，如果我们把公司和行业作

为一个整体来考虑，就能掌握这些关于信任、亲密性和分享的思想，并有效地加以使用，不仅能让工程部门受益，整个行业都将受益。

## 多样性

研究我们的群体从属关系时，必须考虑人们所属的不同群体。很多资料已经很深入地讨论了这个问题，而且我们也没有足够的篇幅全面介绍这个内容，尽管如此，我们还是会简要讨论多样性的好处和维度（坐标轴），并说明团队和组织如何创建包容性的环境，从而对企业以及企业中工作的个人都有好处。

## 多样性的好处

多样性对创新至关重要：来自不同背景的不同想法、视角和观点是研究新想法的一个重要部分。<sup>注4</sup>多样化的团队由于其特有的经历，能开发适合更大客户群体的产品。不同群体或个人合作越紧密，人们就越有创造力。技术行业往往非常单一化，主要由男性组成，也就是说，技术行业中这一类人的比例远远高于普通大众中这类人的比例，这会严重影响其革新和创造力。

优势在于不同，而非相似。

——Stephen Covey

与单一群体相比，异类群体中交换的信息范围更宽，讨论的话题更多。类似的研究表明，性别也有同样的效果：男女混搭的群体与全部由男性组成的群体相比，不论在个人还是群体层次上都有更好的表现。

群体在完成一些需要创造性的任务时，或者需要与这个群体之外的人员交互时，这些优点会更为显著，在这里，发散思维会有很大帮助。在实际中，这意味着如果一个团队需要与客户交互，增加这个团队的多样性会很有好处，这会进一步增加客户的满意度。2000年一项由研究人员和管理学教授Orlando C. Richard领导的研究表明，在公司和企业成长阶段，劳动力的文化多样性会带来更高的公司绩效。<sup>注5</sup>

尽管多样化的群体确实会在个人、团队和公司层次上带来更高的绩效，但是也有缺点，多样化也会导致人际冲突的增加，这在短期内可能会导致士气低落。不同的观

---

注4： Vivian Hunt, Dennis Layton, and Sara Prince. “Why Diversity Matters,” McKinsey.com, February 12, 2016.

注5： Orlando Richard, “Racial Diversity, Business Strategy, and Firm Performance: A Resource-Based View,” Academy of Management Journal 43, no. 2 (2000).

点、预期和观念确实会带来不一致。重要的是需要确保这些不一致得到妥善的处理，而不能看谁嗓门最大就谁说了算。再来考虑devops组合，我们必须知道最终要朝着相同的目标努力，而且要理解尽管存在着分歧和不一致，但我们的目标是相同的。

## 多样性和交叉性坐标轴

技术领域中的很多多样化倡议最初都是因为发现工作场所中缺少女性而发起的。纠正性别差别很有必要，但还不够。

多样性可以有很多不同的坐标轴。这包括：

- 性别和性别表现形式。
- 国籍。
- 性别。
- 年龄。
- 经验。
- 身体状况。
- 信仰。
- 家庭状况。

任何一个坐标轴上增加多样性都很重要，但一个坐标轴并不意味着你的公司确实具有多样性，也不表示这是适合大量不同人群工作的一个安全的场所。交叉性定义为研究不同形式压制或歧视的交叉，以及这些不同形式的压制如何相互关联。这个词最初是法学学者Kimberle Crenshaw<sup>注6</sup>提出的，在考虑你的公司的多样性时，这是需要考虑的一个重要方面。

与所有其他devops实践一样，多样性并不简单，不能一蹴而就，绝对不是一次实现后就能搁置一边不再考虑。这是一个交互的过程，必须进行监控和度量。考虑多样性的原因很重要，你的工作是否成功将取决于这些原因。不论是多样性还是交叉性倡议，都

---

注6： Kimberle Crenshaw, “Demarginalizing the Intersection of Race and Sex: A Black Feminist Critique of Antidiscrimination Doctrine,” Feminist Theory and Antiracist Politics (Chicago: The University of Chicago Legal Forum, 1989)。



应当发自内心地出于改善公司所有人生活以及整个社区的考虑来开展，要创建一个高生产力的环境，同时这个环境要包容可以帮助推动生产力的所有人。

## 无意识偏见

无意识偏见(Unconscious biases)来自于我们的环境和我们所处的时代和文化，通常我们并没有注意到它们的存在。这是一些根深蒂固的思维模式，例如，这会让我们认为男性就是比相同资历的女性更合格，而我们甚至不会意识到自己是这样认为的。要克服无意识偏见，最好的办法就是开始（并一直）注意这些偏见，这也是为什么像Google等公司开始为他们的员工提供无意识偏见培训的原因。

## 招聘注意事项

想要提高人员多样性时，在整个招聘过程中需要考虑很多注意事项。我们会在第五部分介绍扩展团队的其他相关问题，这里先指出几点，确保你的招聘方式与你的组织一样包容：

- 注意可能不当的言辞，如招聘启示和与招聘人员的沟通中过于男性化或军事化的措辞，以及性别偏见的言论。尤其应当对外部招聘人员尽可能提供指导，使他们了解你的公司所要体现的格调和文化。这一章的案例研究还会给出这方面更具体的例子。
- 当心那些无意识偏见。即使我们本来并无恶意，但通常可能会无意识地因为一个人的名字听上去像是男性的名字，就认为这个人的简历更优秀。只要有可能，要让招募和招聘过程中涉及的每一个人都接受无意识偏见培训，尽可能从招募过程中得到第一手信息。
- 有一些招聘人员和顾问非常擅长创建多样化的团队。如果你发现找到你想要的众多多样化应聘者很困难，就很有必要引入一个在这个领域有更多经验的专业人员来提供帮助。
- 尽管有些团队喜欢让应聘者提交“作业”作为筛选过程的一部分，但要记住，这可能会让边缘化群体的成员处于劣势，比如承担家庭责任的女性或者没有时间或不愿意为公司免费工作的人。

## 维持一个包容的环境

你可能想面试和招聘多样化的员工，但是如果无法把他们留下来，这就没有太大意义

了。除了多样性，公司还需要包容性，确保少数群体中的个人能够获得归属感，并得到鼓励保持他们在工作群体中的独特性。

有一种情况很常见，小创业公司在工程团队中聘用了第一位女性。从团队其他人的角度来看，这很好。他们更多样化了，而且现在他们有了一个内部资源，可以帮助他们避免所有可能的性别偏见！不过，从这个女性的角度而言，这可能不算是一个包容的环境。

很多男性团队聘用了第一位女性后，人们进办公室或会议室时可能会说，“嘿，伙计们”，然后看到这个女性，心想这么说她会不会认为自己被排除在外，然后又蹑脚地加上“……和姑娘们”或“……和美女们”。这是好意，但是很多女性认为，加上这个“尾巴”会让她们感觉更难堪，而且更觉得被排除在外而不是融入到团队中。一方面，我们不应该把成年女性称作“姑娘”，除此以外，让人们特别注意到一些人与其他人之间的区别也会增加被排除在外的感觉。

在这种环境中，少数群体中的成员通常还有一些额外负担，因为除了他们正常的工作职责之外，通常还希望他们承担大量与多样性相关的工作。可能会要求他们审查招聘启示，确保其中没有性别偏见语言，或者让他们代表公司，使公司在行业或招募活动中表现出一种多样性。常常会让她们代表所在群体的所有成员，不过，没有哪个女性可以作为所有女性的发言人。技术领域，传统的少数群体也同样如此。

考虑多样性和包容性时，要考虑员工当前可以参加的社会团体和活动。是否允许或鼓励员工创建新的团体？如工程领域的女性团体，以此建立同事间的联系以及创建安全的支持空间。人们能不能成为领导或类似于领导的指导者？例如，如果在工程中女性员工都在低级岗位上，而高级工程师或工程领导岗位上根本没有女性员工，这些低级员工可能就会怀疑他们在公司里是否有发展空间。

要确保一个环境尽可能包容，在这方面，还要考虑办公室活动（特别是社交或“业余”活动）是选择性加入(opt-in)还是选择性退出(opt-out)。选择性加入活动是指除非员工选择参加否则并不参与的一类活动，而选择性退出活动则默认所有员工都参与，除非他们特意选择退出。

看起来选择性加入会建立一个进入门槛，这可能阻碍人们参加，不过选择性退出也存在一个问题，这要求人们承认他们不想做某件事，或者要提供不想做的特定原因。例如，很多公司把喝酒看作是默认的工余活动，创业公司的员工在喝酒方面尤其容易遭受同伴压力，如果一个人下班后不去酒吧，就会给这个人冠以不同类或“不合群”的标签。如果一个新员工必须向他还不太了解的新同事解释为什么不愿意去，这可能会

让他感觉不舒服，而且也会让他感觉被排除在外。在这个例子中，如果有一个办公室小厨房，备有很多酒精和非酒精饮料，不会强制希望“每个人”什么时候一起来喝一杯，这就会是一个更包容的环境。

## 成见威胁

如果人们发现他们所处的位置存在这样一种风险，需要确认对他们自己以及所在群体的一种负面成见，这就会出现成见威胁。在300多项不同的研究中已经发现，这会降低个人绩效，尤其是他们认为根据他们所属的群体或身份可能会受到区别对待时。下面以女性在数学方面弱于男性的成见为例。

受到这种成见影响的女性与没有受这种成见影响的女性相比，前者在数学测验中的表现会比后者差，而且会表现出更多应激反应，如心率加快，皮质醇水平升高。如果长期处于成见威胁之下，与长期压力一样，会对精神和身体健康产生同样的长期负面影响。

研究表明，团体的归属感可以帮助缓解成见威胁。如果人们在更大的团体或环境中受到欢迎，感觉自己真正融入其中，他们就不太会受负面成见的影响，否则负面成见可能会降低他们的绩效（并影响他们的健康）。

要确保你的工作环境尽可能包容，可以有很多做法。考虑组织的多样性和包容性时，要记住以下几点：

- 确保你用的所有招募人员都了解你的多样性和包容性目标。
- 派员工参加无意识偏见培训或联合技能研讨会（Ally Skills Workshop）。
- 以身作则，并叫停有问题的语言或行为，而且不要只是让这成为少数群体成员必须承担的职责。
- 组织建立员工资源团体。建立一些场所来解决不同个人的需要，包括建立社区、网络和支持。这些团体有利于缓解个人因为不同于多数群体所产生的一些负担，使他们能适应环境。
- 审查你的工作环境。确定除了政府强制的要求外，能力不同的员工是否能掌握这个环境中的关键要素。
- 如果要求一些人完成审查工作，即审查招聘启示是否更为包容，那么需要对他们的工作提供相应的报酬。



- 要一直注意你的语言或行为是否存在性别偏见，而不只是当那些群体中的人在场时才注意自己的言行，要预先营造一种包容的气氛。

建立真正包容的环境不仅对这个环境里的个人有好处，对整个组织也有利。如果人们不用处理各种压力源，他们就能更加协作、更乐于沟通而且更有创造性。

## 团队和组织结构

从一个团队到一个更大的组织，个人之间的关系会变得更微妙。英国人类学家Robin Dunbar提出一个理论，指出了可以与一个人维持稳定社会关系的人数限制。通过对猿猴的研究以及对大脑皮层大小及其处理容量的调查，他提出了这个上限，现在称为邓巴数，大约为150个稳定关系。<sup>注7</sup>超出这个规模，群体和组织就必须使用更严格的规则、法则和执行规范来维护有类似内聚力的群体，这正是大公司总是比小公司有更多管理者的原因之一。

如果组织中超过150人，则需要一组不同的文化实践和更严格的规则和规范来维持一个稳定、内聚的群体。不过，组织中的人越多，就需要付出更多的努力来建立群体之间的关系，以确保正常的信息流动和准确的理解。如果这些文化实践限制了组织中个人之间建立关系，这会对整个组织产生巨大的影响。

社会结构影响文化，而文化又会影响社会结构。涵盖协作、合作和亲密性价值观的文化会影响底层的组织结构。与所有改变一样，特别是有关于权力差别，转变为这种文化会带来争议行为。在第10章中，我们会讨论解决这些争议的一种方法。

如果一个组织没有足够重视这些人际关系和群际关系，这会在各种不同的行为方面体现出来。如果不同团队之间有重复的工作，或者在竞争或互相排斥的项目中并行工作的群体之间存在重复工作，这不仅说明团队之间缺乏沟通，也说明缺乏理解。不同群体或项目之间来回重复发出客户ticket，或者责备性事件响应不断增加，这些都反映出信任在减少。

## 找出团队之间的共同点

如何建立信任、创造共同经验并与更多内群体创建更包容的组织？

尝试在不同团队之间搭建桥梁时，遇到的主要困难包括：

---

注7： R. I. M. Dunbar, "Neocortex Size As a Constraint on Group Size in Primates," Journal of Human Evolution 22, no. 6 (1992)。

- 目标的差异。
- 对成功的度量。
- 领导的差异。
- 沟通方式的差异。

不同的团队通常有（至少表面上）不同的目标。尽管有人可能会说每个团队的目标都是帮助公司整体取得成功，但是为了达到这个目的，不同团队采用的方式相互之间往往存在冲突。

在devops领域，这就有一个经典的例子，开发团队的目标通常包括尽快为顾客提供新特性（或bug修正），这与运维团队的目标相悖，运维团队往往以保证正常运行时间之类作为主要目标，要确保所有服务器和服务都可用。这些目标在很多方面都可能冲突。部署需求或过程会设计为尽可能减少部署错误对可用性的影响，这要支持运维的正常运行时间目标，但与开发的快速交付目标不一定一致。在这种情况下，开发人员可能还发现他们与QA工程师意见不一致，QA工程师的目标是找出并修正缺陷，这也会减慢发布周期。

尽管不同团队的既定目标没有直接冲突，但团队度量成功的方式可能确实是冲突的。关键绩效指标（Key performance indicators, KPI）可以用来评价组织的进展和成功，不过如果这些指标与最有利于企业的方面（可以解读为：促使企业取得成功的客户）不能一致，不仅不会帮助提高整体绩效，最后反而会阻碍发展。

如果只是根据完成的部署数来度量开发人员的工作，或者更糟糕的，根据他们编写的代码行数来度量，为了达到这些目标，他们可能会交付更低质量的代码或者可能花费时间完成客户并不想要的特性。

如果根据找到的bug数来度量一个QA团队的工作，他们可能会减慢发布，放弃更快地交付产品，等待找出尽可能多的bug以达到他们的任务要求。这些目标或度量标准是冲突的，不仅如此，最后还会带来影响，与企业的总体目标不相适应。

团队之间以及团队中个人之间可能会有不同的领导和沟通方式，这也会导致团队之间产生隔阂。存在很多不同的沟通和工作方式，这会影响人们之间的交互。如果人们发现自己会抱团形成团队，由于人们下意识地更愿意聘用和他们类似的人，更喜欢与类似的人共事，一段时间后就会发生这种情况，最后就可能不是在个人层次上而是在团队层次上出现沟通或工作方式的冲突。

如果领导人有不同的风格或管理价值观，这可能会带来或增加这些差异。有些管理者更喜欢亲力亲为甚至事无巨细地管理，还有些管理者更多地采用放手的方法，鼓励下属独立尝试，这二者之间意见就不一致。如果一个人更看重生产力而忽视所有其他方面，可能会与更关心人员培养和人际关系的领导人发生冲突。如果组织中不同管理者是否达成共识将决定谁能获得某些项目或得到提升，这些冲突会产生严重的影响。

尽管有上述这些差异，不过也有一些方法可以找出团队和群体之间的共同点，下面几小节将介绍这个内容。

## 从竞争到合作

考虑有不同目标的团队时，我们通常会考虑彼此之间存在直接竞争的团队。能更好地完成目标的团队往往是能得到组织的更多采购预算、更多项目资源或更多人员的团队。给定这些固定的资源，团队如何从竞争转向合作？

在第二部分我们已经提到，如果个人或组织在追求相互对立的東西，就会出现竞争。很多市场中，由于要做到更快、更廉价或更好，或者所有这三者都要做到，这些需求就造成了竞争。这被看作是自由市场的一个必要条件。

符合公平竞争是全世界市场发展的一个基石。这个竞争法则的原则是：禁止限制自由贸易的协定或做法，不允许主导厂商的舞弊行为或导致这种地位的做法，另外会对威胁竞争过程的合并和收购实施监管。在很多方面，竞争有助于激励竞争者更努力地工作，发现更有效或更有创造力的问题解决方案，以及为客户提供更多选择。

少量竞争是有好处的，不过，太多的竞争总体上却会带来更大的有害影响。“公地悲剧”一词常用来描述这种影响，个人的行为是独立的，都只考虑其自己的利益，由于过度消耗一些公有资源，以至于最终与群体整体的利益相悖。这个词是从经济学家 Garrett Hardin<sup>注8</sup>在1968年写的一篇论文的标题而来，原来是描述不加管理地在公有或公共草地上放牧的羊群所产生的影响，因此有了“公地”一词。

这是博弈论的一个经典例子，博弈论研究的是人们在面对涉及多方面、集体行为和互动决策的问题时如何做出理性选择。Florian Diekert是一位人际关系和经济学学者，他在2012年所写的文章“从博弈论角度看公地悲剧”中提出原先对这种情况的讨论并不完全实际。<sup>注9</sup>他指出尽管这种情况下的合作很困难，尤其是当风险变得越来越高

注8： Garrett Hardin, “The Tragedy of the Commons,” Science, December 13, 1968.

注9： Florian Diekert, Florian. “The Tragedy of the Commons from a Game-Theoretic Perspective,” Sustainability 4, no. 8 (2012)。



时，但无限制的个人自由带来的悲剧并不是不可避免的。

实际上，美国政治经济学家和诺贝尔奖获得者Elinor Ostrom明确了在这些公地情形中对合作成功至关重要的几个因素。<sup>注10</sup>她强调，要保证合作成功，必须有：

- 一种控制整个群体中从属关系的方法。
- 社交网络。
- 可观察所涉及的每个人的行为。
- 对个人的分级处罚。
- 一个不过度变化的资源。

Ostrom和其他博弈论研究人员指出了处罚行为的作用对整体合作有显著的影响。非合作行为受到集体惩罚时，最后合作行为就会变成自发行为。例如，如果一个团队中每个人都指出或者不能容忍有人打断别人讲话，这个团队就会共同认为打断别人并不是一种让人们听到你的声音的有效方法，这样一来，打断别人的行为就会越来越少。不过，如果有人允许打断别人，就不会看到这种自发的行为了。



搭便车是指有些人只攫取社区提供的好处而不做任何投入。这是有害的，会阻碍其他人的参与或贡献。这方面的一个例子是“喜欢提问”，这种人总是问问题，而不考虑所问问题的主旨或者问问题的频率，而当他们的期望未能得到满足时，就会很无礼。类似地，如果一个群体中没有处罚，或者不真正实施处罚，这也会影响个人贡献。博弈论显示，如果对非合作的行为没有任何反应，人们就会很自然地越来越多选择非合作行为，以保护他们自己的个人利益，因为整个社区不能帮助他们实现这些利益。

因此可以得出，要创建合作的工作社区而不是竞争的社区，必须确保在某种程度上体现Ostrom描述的这些因素。群体从属关系由招聘过程来控制，社交网络以组织和团队层次结构的形式存在，另外还包括非正式和正式的社交。大多数人和团队的行为在某种程度上都是可观察的，而且存在很多不同的处罚，如绩效提升计划、撤销不需要的的项目，缩减预算以及裁员或解雇。

最后一个要素很有意思，即不过度变化的资源。特别是在努力占稳脚跟的小创业公司

---

注10：Elinor Ostrom, *Governing the Commons: The Evolution of Institutions for Collective Action* (Cambridge, UK: Cambridge University Press, 1990)。

里，或者在面对衰退和裁员的较大公司中，可以观察到，与较稳定的工作环境相比，这些变化较大的环境中个人和团队行为会更有竞争性，而更少合作。不论组织的规模如何，你在考虑如何建立和维护一个持续的、合作的工作环境时都要谨记这些要素。

## 建立团队同理心

有了同理心，运维工程师就能认可快速、频繁地推送代码的重要性，而不会大惊小怪。开发人员也能理解编写太过臃肿、太慢或不安全的代码会带来问题。同理心会让软件开发者和运维人员相互帮助，从客户角度考虑，为他们提供最好的功能和操作性能。<sup>注11</sup>

——Jeff Sussna

devops运动的重要原则之一就是培养同理心，不仅是对“隐喻墙”对面的人或公司的其他部门建立同理心，还包括客户。同理心和理解是紧密相关的，如果没有深入地理解你的客户想要什么和你要为他们解决哪些问题（以及这些问题是否确实是真正要解决的问题），你的企业不太可能取得成功。

## 指派运维人员

多年来，运维工程师和系统管理员总被看作与公司其他部门格格不入，他们往往脾气暴躁，不好相处，看不起周围的人，对所有事情都喜欢说“不”。在实际中当然有很多这样的例子，但组织要停止这些有害的行为，打破对他们的这些看法，这对于组织取得成功非常重要。对于现代企业尤其如此，在这些企业中，IT和运维对产品的成功至关重要。原先系统管理员可能一个人待在一个服务器室里，连着一个只提供电子邮件和打印机服务的网络，不过这样的日子已经一去不复返了。

在严重割据的环境中，其他团队一直在努力，希望从运维团队得到他们需要的服务和支持。有些IT部门设置有一些不必要的过程来为其他人提供帮助，这可能是出于历史原因，也可能是为了减少看起来浪费时间的请求（比如，一个用户每个月至少需要重置一次密码，这就是一个常见的例子）。有些部门铺得太细，缺少足够的资源来完成自己的工作，更不用说为其他团队提供真正的帮助。如果基础设施不稳定，会导致大量时间都花费在应急的救火上，而无力去完成主动的工作。

要改善这种情况，组织可以考虑为其他团队指派运维工程师。如果团队可能需要大量运维支持，可以有一个指定的运维团队联系人。例如，一个需要完成大量监控、性能

---

注11：Jeff Sussna, “Empathy: The Essence of devops,” *Engineering.IT*, January 11, 2014.



优化和容量规划的Web或API开发团队就需要很多运维支持。指派一个人到这个开发团队提供支持，这样开发团队就会有一个一致的联络点，而不是不断地新建客户支持ticket，随机指派一个不同的人（如果是这样，就必须重新解释他们的问题、上下文以及他们已经尝试过的做法等），而且还可以为运维团队的成员提供一致性。为团队指派的运维工程师会全程参加他们的团队会议或站会，保证随时了解可能需要额外运维支持的问题，如开发的新API端点可能会对API集群增加额外的负载。如果这需要新硬件，让运维人员更早了解这些开发工作对两个团队都有好处。

为了有效地使用这个方法，需要谨记几个要点：

### 指派而不是全职

要记住最重要的问题之一是运维工程师只是指派给这个团队，而不是为这个团队全职工作。这意味着，要理解运维团队还有自己的工作，运维团队的成员不能（也不应该）把100%的时间都用来帮助其他团队完成工作。这也意味着为一个给定团队指派的运维工程师不必负责一个人单枪匹马地完成这个团队所有与运维相关的工作，事实上，他们只需要作为这个团队的主要联络点，通常只是管理或监督相关的工作。时间和项目管理技能对于运维人员开展这些工作非常重要，所以要确保他们在这方面有必要的支持和培训。

### 运维团队规模

除非你的公司规模很小，只有一个小型或中型团队完成工程，否则如果没有一个适当规模的运维团队，将无法很好地开展工作。正常项目和职责之外的工作会加大压力和负担，所以团队要配备足够的人员来保障这一点，这至关重要。在实践devops方法的Etsy，有大约15个运维工程师支持数百个其他工程师，这绝对不是一个只有2到3个人的运维团队能够做到的。



另外要记住，并不是所有指派的运维工程师都有同样的任务。不同的团队中，所指派的工程师工作量是不同的，有些几乎无需做任何具体工作，只需要在一旁监督与运维相关的事情（如果团队对运维知之甚少，他们甚至不清楚自己不知道什么，在这种团队中这种情况就很常见，与之不同的是另外一些团队，他们知道什么时候需要寻求运维帮助），有些则要求运维工程师做大量工作。可以考虑根据项目而不是团队来确定运维工程师的指派，来帮助均匀地分摊工作量。

### 好坏方面都包含

一直以来，运维都是一个不讨喜的职业。如果一切正常，运维工作几乎完全是不



可见的，只有出问题时，大家才会注意到运维。运维团队和成员工作得很好时，他们很容易被忽视，而一旦出现问题或状况，往往会被大家指责诟病，特别是在问责环境中。要把指派的运维工程师也纳入到团队好的方面或有趣的活动中，而不是只在你需要帮助时才和他们讲话，这样有助于建立共情的关系。这可能很简单，只需要邀请他们参加团队午餐或晚宴，或者在团队外出活动时让他们加入，一点点包容会有巨大的意义。

## 双向的学习

在团队之间建立这种关系的好处之一是可以促进学习，原本在分享知识和经验方面没有太多共同点的人们之间也可以相互学习。如果一个团队请指派的运维工程师参加他们的会议，他提出诸如如何监控、这对于“生产阶段”有什么意义、如何做出故障和灾难恢复计划等等问题，一段时间后团队不仅能逐步理解这些问题的意义，理想情况下，他们还会主动地考虑这些问题。不过，为了充分利用这种指派运维人员的安排，这种学习需要在两个方向上进行：运维人员会对其他团队所做的工作有更深入的理解和认识。他们会意识到这些团队人员的出发点和努力，不再把他们看作是不见真人的人名或email地址，而原先在他们的印象里这些人只是在需要什么的时候才会来找他们。他们可以把这种理解和同理心带回给运维团队的其他人，逐步消除壁垒，扩大他们所认为的内群体范围。



除了运维，这种指派安排对其他团队也有好处。例如，指派的设计或用户体验团队可以帮助确保所创建的产品使用起来简单方便，这对于内部工具团队很有意义，这个团队需要创建整个组织使用的工具。如果这个团队主要由具有后台或系统管理经验的人员组成（通常都是这种情况），倘若参考他们自己的设备，他们很可能会提供一个命令行工具，这对他们自己很方便，但是对于不太习惯命令行的其他人来说就不合适了。如果有一个指派的设计人员或用户体验工程师，就可以帮助他们实现一个适合更多人的工具。在安全领域中，让指派的工程师与其他团队合作也会产生显著的积极影响，因为与运维类似，人们对安全的认识往往是这不是最紧迫的但在产品的整个生命期中都要考虑，不过事实上总是在最后一分钟才会考虑安全。

## 训练营和岗位轮换

指派运维人员等项目允许人们与其他团队更多地交互，而其他项目，如训练营和岗位轮换，则允许人们暂时加入其他一些团队，通过这个途径来扩展其知识和技能，同时还可以增加同理心。

训练营（bootcamp）一词描述了员工在一个给定团队的任期刚开始时所采用的一种方

法，员工刚加入这个团队时，需要在其他团队（而不是他自己的团队）工作一周或数周。人们主要会在与其团队定期紧密合作的团队中采用这种训练营方法，所以一个开发人员可能在运维团队“训练”一周，另一个开发人员在转入正常工作之前可能先加入安全团队“训练”。在最开始时这样做的好处是，人们不会处于某些项目的中间阶段，那些项目中的其他人不会依赖他们，相应地对他们的时间限制会少一些。另外，他们对于其他团队或“本该怎么做”没有太多先入为主的想法，所以他们会用全新的视角看问题，这通常会为团队如何协作带来新的想法，否则可能很难做到。

一个人在特定岗位的任期开始后所采用的方法通常称为岗位轮换。有些组织或团队通过合理安排，使公司能够做到每年完成高级岗位轮换，这些团队会充分规划，使得部分人员几周不在原岗位也不会有太大的影响。在选择暂时加入哪些团队工作时，高级岗位轮换可能更灵活一些。因为他们现在对自己的团队以及定期合作的团队都很熟悉，所以可能会利用这个机会在另一个完全不同的团队工作，如一个运维工程师可能会加入移动开发或前端工程团队，来探索他们原本没有机会接触到的不同领域和技术。

总的来讲，作为另一个团队的成员与指派人员有同样的好处，不过通常可以为培养同理心和理解建立更深厚的基础，而且还会使不同的群体从属关系有更多重叠。如果一个组织中较难做到这种规模的训练营或岗位轮换，可以利用其他方法在更小规模上得到类似的结果。可以为不同团队的人员提供机会，让他们相互结对工作来完成项目，这会在几个小时而不是几周内就开始建立同样的联接。例如，你可以建立一个选择性加入系统，工程师们根据bug或支持ticket清单结对工作，这个系统可以面向内部产品，也可以面向公司使用或维护的开源项目。

## 非工程的岗位轮换

前面已经提到，这本书中描述的原则不只是适用于开发人员和运维工程师创建和维护更强大的公司和行业文化。很多技术公司，尤其是创业公司，往往非常看重工程师，但是这常常有一个代价，就是会忽略非工程团队。工程师可能会发品牌工作衫，参加会议的行程费用全免，其他部门很自然地就会认为自己不被重视和不被认可。

devops运动的驱动要素之一是对运维工程师、系统管理员和一般意义上的IT部门给予更多的理解、认可和同理心，帮助公司的其他部门了解这些领域提供的价值以及将部署扔到墙对面的运维团队等行为（“开发时一切都正常，现在只是运维的问题”）对他们会有什么影响。运维领域被忽略或被错误对待已经太长时间了。很多地方这种情况正在发生变化，不过有一点很重要，有类似境遇的其他团队可能还没有得到我们的重视，他们的情况还没有改观。

在很多公司，客户支持团队就属于这一类。他们没有工程岗位所拥有的声望，但客户支持人员是你的公司面对产品使用者的“脸面”。他们往往要忍受着人们不满意时的抱怨，就像网络不可用或者打印机无法打印时系统管理员所面对的境遇一样。作为一个产品用户，如果你打电话或写信给客户支持人员，几乎可以肯定你是因为对某个方面不满意才会联系他，而且很遗憾，出于人类本能，我们常常会迁怒于和我们直接对话的人，尽管我们知道他们并不是直接导致这个问题的“罪魁祸首”。支持团队的流动率往往比其他团队高得多，而且常常感到在公司不被认可或被轻视，尽管他们的角色至关重要。

支持岗位轮换可以帮助缓解这个问题，让其他团队的人偶尔花几小时回复支持email或者接听客户的抱怨。大多数情况下会收到大量相同类型的问题，利用完善的文档或预建的回答模板，人们只需要很少的上岗准备，就能帮助解决这部分支持问题。当然，解决更困难和不太常见的支持问题还需要相应的技能和耐心，并不是说这些技能和耐心不重要，这里要强调的是，这是一种让其他群体（通常是工程师）帮助分担支持工作的方法，同时更重要的，采用这种方法还可以让他们对客户可能有哪些问题有所认识（有可能与开发人员原先想象的问题完全不一样），另外还能了解到支持团队对公司的价值以及这个工作有多艰辛。

除了允许和鼓励在自己公司的不同岗位之间转换和轮换，一些组织开始在不同公司之间采用同样的原则。这通常称为工程交换，不同公司中职位大致相当的两个工程师交换岗位2~3周，可以同样地实现知识交换、经验共享并建立同理心，不过是在更大的范围内。公司之间不再抱着之前数十年那种竞争的观念（“不惜一切代价保护秘密武器”），而允许员工通过会议讨论和开源软件等方式分享信息，知道这样做并不会削弱公司的实力，而是会加强整个行业。

当然，这种轮换项目是否成功取决于相互分享和学习。如果一个公司允许他们的交换工程师作为另一个公司全职的团队成員，参与具体的工作，而对方什么都不做，只是查阅文档，这就是一种单边关系。回顾之前讨论的公地悲剧和维持合作行为的要素，对这些非合作行为的处罚可能是公司将来会拒绝再与这些“差劲的”组织开展交换项目，减少那些“坏家伙”能够拿到的信息。

如果组织希望增进其整体以及团队间的同理心，不仅要查看不同团队和群体之间如何交互，还要查看这些不同群体的价值，这包括实际价值和所认为的价值，这很重要。传统上，类似IT和支持等职能部门通常被看作是成本中心，它们不直接增加利润，而是会增加运营公司的货币成本。尽管devops运动已经开始改善人们对运维团队整体的认识，但在这方面还有很长的路要走。成本中心的员工通常工作都不讨好，他们的流



动率更高，薪酬更低，而且人员精减时他们的缩编风险也更大，不过尽管在公司人才阶梯上他们处于底层，但是如果有适当的人员、培训和资源，他们会在客户忠诚度或基础设施稳定性等领域带来巨大的收益。

扩展我们的视角来考虑这样一些团队以及他们如何影响整个企业时，我们相信，不再按公司层次结构这样一个梯子来考虑会很有帮助。这个梯子是一个过于简化的模型，不仅会导致人们对职业发展的误解（正如我们在前面提到的，从工程岗位转向管理只是一个职业调整，而不是提升），而且如果按这个梯子来考虑，人们更难把自己放在“低于”其他人的位置上来理解其他群体。另外这还会忽略群体如何相互联系和相互依赖。

想象一下，如果不作为一个梯子来考虑，而作为一个绳索金字塔。有时我们会在运动场上看到，拉紧的彩色绳子会提供足够的拉力，可以让孩子们攀爬和游戏（见图9-1）。尽管这仍然是一种层次结构，但是在这个模型中，可以更容易地将IT和支持等角色想象成这个金字塔其他部分的基础，而且可以认识到，如果没有一个强有力的基础，这个结构整体的稳定性和成功都会受到威胁。梯子可以没有下面的一级，它仍然可用，正因如此，这些领域通常被看作是成本中心，也是最先考虑裁员的部门，但必须考虑他们带来的价值，尽管不是货币价值。如果你切断这个金字塔底层的某个绳索，整个金字塔都将失去稳定性。

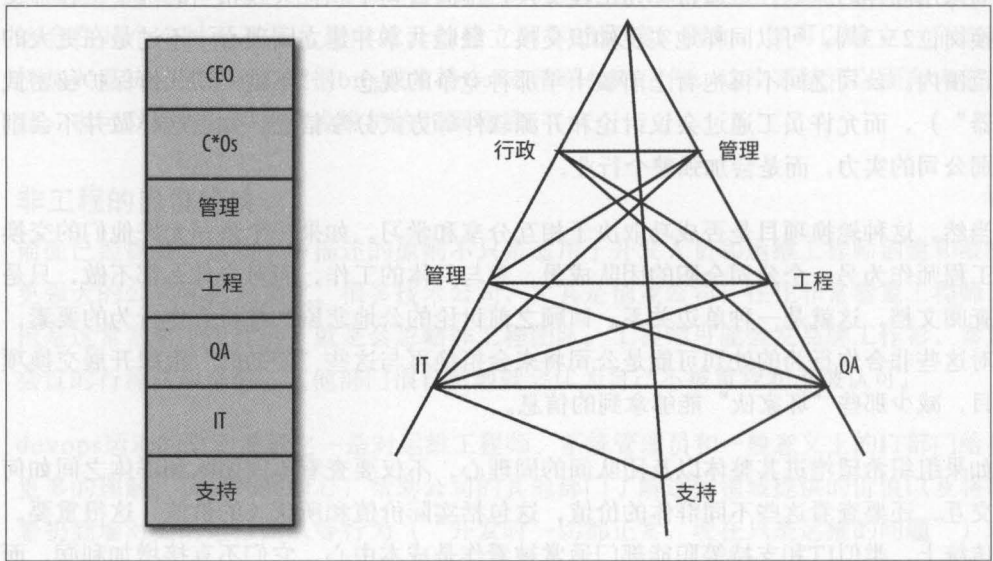


图9-1：梯子与金字塔层次结构

## 改善团队沟通

团队相互之间除了要有同理心，还需要能够有效地沟通，从而能真正地合作。第二部分对沟通方式、媒介和协商类型做了讨论，在此基础上，这一节将分析如何在团队层次上应用这些概念。这是很重要的一部分，有了这一点，devops文化才会从只针对少量团队或工程部门的文化转变为针对整个组织或企业的文化，即不只是少量团队或工程部门能从中受益，整个组织或企业都将受益。

在最简单的层次上，群体的沟通会基于并放大群体成员的单独沟通，而不论好坏。群体可能采用不同的沟通方式，不论哪一种沟通方式，往往都是最强势的人胜出。通常这意味着说话声音最大、讲话时间最长或者打断别人最多的人会主导讨论，并在争论中“取胜”，不一定是因为他们的想法正确，而只是因为不好争辩或不愿意打断别人的人会更快地退出这些讨论。一个在团队中很强势、经常打断别人的人会改变整个团队的沟通状态，因为其他人也会开始越来越多地打断别人，想要插上话。

群体还会以他们的领导人为表率来确定他们要如何沟通，这可能是一个管理者，也可能是群体中很有权威或受到大家尊重的人。如果领导人不能很好地沟通，不能充分地沟通，或者他们的沟通方式不利于一个合作的工作环境，这会对他们身边其他人的沟通方式产生类似的影响，所以要留心你的组织中领导人所设定的标准。

在很多组织中，通常会有很多关于沟通程度以及通过哪种媒介进行沟通的讨论。正如我们在前面看到的，从受众范围、紧急程度、上下文以及其他因素等方面考虑，不同媒介分别适用于不同的沟通类型。这些不同的因素再加上个人的喜好，通常很难设置让所有人都满意的原则，不过一般来讲，过度沟通总比缺乏沟通要好。

有些媒介允许人们很容易地设置过滤器来满足他们的喜好，如email客户端可以自动将不同邮件过滤到不同的文件夹，聊天程序允许定制警告词，对于这些媒介，更能体现出过度沟通要胜过缺少沟通。尽管这不是一种十全十美的解决方案，但这允许一定程度的个人定制，避免人们有一种信息被有意压制的印象，否则会助长信息割据和内群体/外群体观念。

另外还要记住组织中个人和团队的分布。随着团队开始分布到不同的位置和时区，让每个人都能面对面沟通会越来越困难。要养成习惯，默认采用适合远程的方式沟通，如email和文本聊天，这不仅可以提供可搜索的沟通内容记录，以便将来参考，还可以让远程工作的人员和团队没有被隔离在外的感觉。

## 发生危机时的沟通

有些情况下沟通很可能会出现一些问题，比如发生类似网站瘫痪等危机时。尽管这些特定情况下的沟通是一个相对新的研究领域，但从更一般的意义上说，涉及多人和多个团队的危机中的有效沟通并不是一个新概念。尤其是医疗领域已经在这方面做了多年的研究，这个学科中很多策略已经过实际检验，也可以应用到我们的领域：

### 跨部门沟通

人们通常习惯于只与他们自己的团队或部门的成员沟通，在涉及多个团队或学科共同排查或鉴定的情况下这可能会带来问题。在这些高压领域之外也可以采用跨部门沟通，这会帮助养成好习惯。医疗团队会与医生、护士和药剂师一起查房。在一个工程组织中，前面介绍的指派运维人员或类似的其他项目也有同样的好处。由于各个团队可能已经发展形成了不同的沟通方式，努力建立和维护共同的沟通标准（如共享不同情况下使用的检查表或模板）也可以减少跨部门沟通中的摩擦。

### 自主性沟通

危机往往不会留多少时间来解决沟通不足或猜疑等问题，所以人们要适应自主性沟通，这至关重要。不过，直接并不意味着粗鲁。在这个领域，要确保人们能受到培训，帮助他们直接而有效地沟通，这会很有帮助。有时文化差异会对自主性或直接沟通造成障碍，比如，不同的人群（如按性别或种族划分的人群）会采用不同的方式沟通，这就会影响更大范围的沟通。问责文化通常也会导致人们无法直接沟通，所以人们要熟练采用无问责和非语言沟通方式，这很重要。

### 特定的批判性语言技术

多年来，已经发展和研究了很多技术可以让人们在高压情况下更容易地表达担心或提出批评。两次挑战规则（two-challenge rule）建议人们对有疑问的方面最多挑战两次。出现危机的情况下，如果只提出一次肯定会遗漏某些方面，特别是有大量工作都在进行时，但是挑战超过两次就可能导致意见不一致或争论太长时间，以至于无法及时采取重要行动。

与正常情况相比，危机期间的风险会更高，所以要确保人们能放心地提出顾虑或疑问，这一点至关重要。这方面有一种技术，称为CUS，<sup>注12</sup>鼓励人们如果存在以下情况就要指出问题：

---

注12：“Pathways for Patient Safety: Working as a Team,” Health Research and Educational Trust, 2008, <http://bit.ly/pathways-for-safety>。



- 对某个方面存在担心。
- 对某个方面不确定。
- 安全可能成为问题。

另外还有一个技术，称为SBAR，其中提供了一些原则，确保能够以建设性的方式表达担心或提出批评。人们应当能够对以下方面进行沟通：

- 场景信息，描述发生了什么。
- 背景信息或上下文。
- 对所认为的问题的一个评估。
- 下一步的建议。

要经常采用这些不同的技术来保证团队形成彼此包容的习惯，并尽可能直接、有效地沟通，不论是在危机期间还是日常的工作当中。

我们已经花了一些时间分析如何提高亲密性，下面再来看在真实世界里亲密性会对组织带来哪些好处。

## 案例研究：某国专利商标局

为了对政府组织中的devops变革有所认识，我们与Tina Donbeck进行了交流，她是某国专利商标局(USPTO)首席信息官办公室系统配置和交付自动化部门主任。<sup>注13</sup>这个办公室（即首席信息官办公室，OCIO）要确保客户所用的技术和工具可以正常使用，保证他们能够对某国公众提交的专利和商标文档完成日常的审查、批准和裁定。

### 背景与指南

Donbeck不仅是这个部门的主任，她还是海军陆战队的一名退伍军人，另外她也是在组织中大力推行devops的一位热心者。她拥有心理学和组织发展专业的学位，从海军退役后，她还为海军的IT人力发展提供了很多年的支持，担任过信息系统安全官和计划/项目经理等职。她指出，到目前为止，她的职业生涯中有一个共同点，就是持续不断的提升：“我很喜欢发现不完善的流程，明确什么能够激发人们，推动他们前进。”

注13： All the views expressed in this case study are Donbeck's own and do not necessarily represent those of the US government.

她目前领导的团队开发、实现并负责操作当前的持续交付平台（USPTO的下一代系统就在这个平台上开发）。这个平台涵盖这个组织软件开发生命周期中的几乎所有方面，因此她的团队与这个生命周期中涉及的很多其他团队都有非常紧密的合作，最常合作的是平台服务决策团队、使用RedHat CloudForms的团队以及软件开发团队本身。

尽管他们确实大量使用了CloudForms，即RedHat的云管理和发布自动化平台，但这个组织主要选择使用开源的工具，以避免庞大的专利使用权转让协定或者必须绑定到某一家供应商。这些工具包括Subversion（提供源代码控制）、Jenkins（持续集成服务）、Sonar（项目和质量管理）、Nexus（仓库管理），另外使用Puppet和Ansible完成配置管理。很多团队发现，这些产品有很大的用户群体，这也是这些产品提供的价值中很重要的一部分，因为这些产品用户不仅能够回答问题和提供支持，而且还在不断开发新的特性和部件。

对Donbeck来说，devops意味着要在技术层面有效而高效地完成持续交付，另外在文化层面上，要基于信任和一种协作精神协调各个不同团队生产高质量的产品。她的主要工作就是要开发和维护高质量的软件产品以及破除对此有影响的障碍，这也是她着力推动devops发展的主要动力。

## 鼓励协作和亲密性

努力建立团队间的信任和协作环境是一个持续不断的过程，这会受到整个组织中多方面的影响。Donbeck认为成功的协作与合作的目标是人们“能够朝着一个共同的目标协同工作，而不需要标定各自的领地（这是我的，那是你的）；协作应当是开放的，不会心存畏惧；不用担心失败或犯错误。”

要鼓励个人和团队协作工作，并寻求对他们的工作以及工作有效性的反馈。“我们利用部门网站上的一个特性请求功能寻求用户群体的反馈，我们会在这里描述和分析请求，如果合适还会列出我们的待办工作。我们还会召开一些午餐讨论会和信息研讨会，积极寻求用户群体的反馈”。代码审查是每个发布周期中必不可少的一部分，可以鼓励个人之间的协作和频繁的反馈，另外可以为团队的新员工指派一个同事向他们讲解应注意的事项。

有些流程比较严格，如必要的代码审查，不过Donbeck指出，不仅允许用不同的工具和解决方案做试验，而且要积极鼓励这样做。“我们的团队喜欢做各种尝试，我们有一个沙箱，可以在这里尝试不同的工具、部件等。如果我们希望把某个工具/产品推广到更大的用户群体，会通过我们的企业架构审查流程来确保这个工具/产品满足必

须遵循的所有管理和安全需求”。这样就能支持灵活性和创新性，同时还能满足作为一个政府机构应当满足的所有需求。

Donbeck在建立社区方面有很多成功的经验，包括加强内部社区联系，以及与外部组织协作和分享。她的团队把会议室变成了团队“DevOps工作间”，在这里“经常会看到超过25个人……在共同完成一项工作，包括开发人员、测试人员和平台支持人员。看到这个工作间表现出的旺盛活力确实非常棒”。这种设置使人们能积极协作，而且鼓励紧密合作的其他团队来访，而不是只依赖于开会。她鼓励这些团队朝着一个公认的最终目标努力（在这里，就是更快、更高质量的部署），并且相互之间要建立信任和尊重。要有这样一个社区，可以安全地讨论和提出不同的观点而不必有所顾虑，这是建立这种环境的一个重要的决定性因素。

USPTO是第一个主办devops会议的联邦机构，第一次会议有超过100位与会者。他们的组织还设立了一个行业日，devops领域的众多供应商受邀来分享他们的产品、想法和最佳实践。这个活动同样高朋满座，有超过100家公司分享了他们的想法。他们还主办和资助了devopsdays DC 2015大会，政府和行业组织都有参加。推荐整个组织的员工参加这些活动，以及关于技术和软技能的其他培训，以此鼓励协作发展和成长。

## 平衡多种观点

与所有大型组织一样，要集成多种观点和工作方式形成一个有凝聚力的组织策略，这需要经过一番努力才有可能做到。有时，不同的人对devops会有不同的解释，所以，对于这样一个问题：“一个成熟或成功的devops组织是怎样的？”会有很多不同的想法。Donbeck指出，“开发人员会处于压力之下，要在进度很紧张的条件下生产高质量的产品，运维人员则往往有另外的困扰，一方面要保持日常运维，保证工作正常进行，另一方面还要创造他们的服务产品，运维人员需要平衡这两方面，所以对于开发人员和运维人员来说，什么是成功会有很大不同。对此有一种共识，我们需要对一些既定的成功准则有统一的理解，另外只要主要目标达成一致，各个职能领域对成功完全可以有不同的定义”。

如果感觉组织中不同的人或团队在朝着不同的方向努力，或者甚至彼此的工作相互对立，这肯定很让人烦恼，不过，实际上能够认识和讨论这些差异正是解决这些差异的关键。特别是在较大的组织中，人们的期望和观点往往不同，如果有一个高层领导团队（就像USPTO的领导团队）能够交流讨论成功是什么，并推动不同的团队和领域形成一个共同的文化，这是成功实现文化转变的一个必不可少的步骤。

有些人可能比较抗拒个人层次上的改变，有些人和团队可能由于之前不太开放的问



文化而观点不一，尽管如此，所有层次上都可以努力克服这种问责意识，消除角色和职责演变所带来的困惑，创建更开放、更协作的文化。例如，CIO可以出去买一些称为“DevOps行动者纪念品”的小玩意儿，发给组织中具体行动来建立这种devops文化的人。

组织中有些部门看起来比其他部门改变更快，尽管有时可能比较慢，不过整体来讲有一种稳步前进的感觉。一方面得到了组织管理层和领导层的支持和认可，再加上日常工作中相关团队的共同努力，这二者结合在一起，使Donbeck和她的组织整体工作很顺利，这也是其他大型组织可以学习借鉴的一点。

不论组织规模、发展速度或复杂性如何，都要认识到至关重要的一点：需要有共同的愿景、目标和成功原则，才能成功地实现各种重要的文化或技术变革。这又回到了我们在第1章介绍的devops组合的概念。如果没有取得共识，就不太可能有效地完成任何改变，或者改变也不会长久。尽管可能需要一定时间才能全面地建立共识，但起码要认识到需要相互理解，而且要为之努力，这是真正建立共识的关键的第一步。

## 改善亲密性的好处

在个人、团队以及组织层次上改善团队之间的亲密性有很多好处。由于同理心和沟通的增加，这会让人们感觉到有人倾听他们的声音，理解他们的问题，这会形成一个提高士气和生产力的良性循环。更好的群体行为会建立更强健、更有生产力的组织，能提供更有创造力的解决方案，在企业各部门之间更有效地协作，并且更快地迭代。改善的亲密性不仅有利于在企业工作的人，对企业本身也很有好处。在这一节中，我们将分析为什么是这样以及如何实现。

## 缩短周期时间

周期时间（cycle time）和交付时间（lead time）都是对工作和生产力的度量，它们都源于看板系统，这是20世纪50年代丰田开发的一个用于精益生产的调度系统。<sup>注14</sup>交付时间是从提出请求到交付最终结果的时间，或者从客户的角度来看，就是完成某个产品所花费的时间。周期时间也是截止到交付为止，不过是从针对请求正式开展工作开始算起，而不是从收到请求时开始。周期时间更多地是度量完成率或一个系统总体的工作能力，如果提出请求后一段时间内没有任何进展或工作，这部分时间就浪费了，周期时间越短，意味着浪费的时间越少。

---

注14：Taiichi Ohno, *Toyota Production System—Beyond Large-Scale production* (Portland, OR: Productivity Press, 1988)。

尽管周期时间是使用看板系统的人常用的一个词，不过并不是所有人都毫无异议地认为这个词能准确表达它通常所要描述的概念。取决于上下文，这个词可能有两个不同的含义。第一个就是上面描述的含义，而第二个含义来自于精益制造，表示两个部件相继离开工作或制造流程的平均间隔时间。出于这个原因，很多看板专家开始把第一个定义称为涌流时间（flow time）以避免混淆。

当一个人自主地去完成一件有难度而且有意义的事情而让身心得到最大程度的延展时，这往往是最美妙的时刻。最佳体验就是做我们喜欢做的事情。

——米哈里·契克森米哈

涌流（Flow）是讨论工作和生产力的度量时经常提到的一个概念。社会理论家米哈里·契克森米哈（Mihaly Csikszentmihalyi）引入了涌流的概念，来描述一个人在参与一个活动的过程中完全投入、精力充沛并沉浸其中的心理状态。契克森米哈根据多年的研究，指出了涌流区别于其他心理状态的6个因素。<sup>注15</sup>这些因素包括：

- 高度集中注意力。
- 行为和意识相融合。
- 自我意识消失。
- 对情况的个人控制或行动。
- 主观时间感被扭曲。
- 内在回报。

个人层次上的涌流是一种意识形态，在这种意识形态下，一个人感觉完全专注于一项需要准备和实践的活动，特别是需要创造力的活动。在涌流状态中，人们会感觉自己很强、很机敏，处于能力巅峰。

团队中也会出现涌流。团队中的涌流与个人的涌流有所不同。观察涌流的一个很好的例子就是管弦乐队，乐队由演奏人员和不同的乐器组成，大家汇集在一起可以创造个人无法创造的声音。团队层次上的涌流比个人层次上的涌流更强。当人们作为一个整体时，会预测彼此的行为，这会带来创造力、生产力和高绩效。团队涌流取决于团队中的每一个人同等程度地参与。强势或自大的人可能会毁掉团队涌流。

---

注15：Mihaly Csikszentmihalyi, Flow: The Psychology of Optimal Experience (New York: Harper Perennial, 2008)。

契克森米哈还总结了促进团队涌流的一些特点，这包括共同的团队关注点、工作的可视化、并行组织工作、工作场所的空间管理，以及（可能也是最重要的）要把团队成员之间的差别看作是机会而不是障碍。我们将在第四部分更深入地讨论工作可视化，以及如何利用工作可视化提供更有组织的工作环境。总的来说，团队涌流的这些特点与有更短周期时间或涌流时间、更少浪费和更大生产力的团队密切相关。

## 消除沟通障碍

缺乏沟通、存在误解以及其他形式的沟通不畅是工作中最大的隐含成本。如果不能清楚地沟通期望，不论是不清楚客户想要什么，不明确一个内部解决方案需要什么，还是不了解人员方面的期望（如希望人们如何协作），通常都会导致这些期望无法实现。如果一开始没有明确期望，或者很多人在不知情的情况下重复彼此的工作，这些重复工作代价也很大。沟通方面的问题会带来时间、工作和金钱的浪费。

提高不同团队之间的亲密性和信任可以鼓励人们更直接、更诚实地沟通。如果人们不能自然地直接沟通他们的问题和顾虑，一个常见的反应就是会变得消极对抗。如果两个团队共同负责某个基础设施，对于谁应当在位来完成这个基础设施的基本维护任务没有达成一致意见，却没有正面直接讨论这个问题，他们看到客户支持ticket时，可能会直接分派给另一个团队而不做任何沟通。每个团队都奇怪为什么另一个团队不想做他们该做的工作，认为将ticket分派给另一个团队就足以给出提示，很快这两个团队之间就会互相不满并失去信任。

如果没有足够的信任和同理心，可能会把类似这样的沟通问题误以为是技术问题。有些人不理解开放和直接沟通的意义，他们对这种问题提出的建议很可能是一些技术解决方案，如一个新的监控解决方案，或者采用一种不同的方法处理支持ticket，而事实上这里的问题只是消极对抗行为和避免所有直接冲突。有时人们会避免直接面对，因为感觉这太过情绪化，但是如果让愤怒和不满继续蔓延，一味拖延问题，这会比一个简短的会面或者直接讨论糟糕得多，因为后者可以很快把问题说开并解决双方的不一致。

人们相互之间了解得更多，就能更准确地解释和理解其他人的想法。正是因为这个原因，远程工作的员工加入一个新团队时，最开始几周可以让他们先在办公室工作，或者尽可能多地与其他团队成员一起工作，使他们能掌握情况，了解人们交流时的肢体语言和语气，在远程沟通之前先了解他们的同事，这是非常有意义的，毕竟在远程沟通中这些非语言线索大多都无法体现。如果远程或分布式团队能很容易地视频或至少音频聊天，这也很有意义。尽管分布式团队很有好处，不过我们还需要考虑到距离和技术对这些沟通和联系产生的影响。





技术本身不会创建或加强联系。Twitter、Facebook、IRC或Slack等服务使人们能相互联系，但是每个人必须培养这些弱联系，来加强他们建立的关联。这些服务的特性会受限於建立服务的团队，所以更多样化的团队创建的产品会得到更大范围的理解，能够克服更多的沟通障碍。不过，尽管这些沟通工具有助于加强联系，但是具体效果完全取决于使用这些工具的人以及他们如何使用。

## 信任

相比于信任度低的组织，信任度高的组织有很多优点。相互信任的同事协作会更多，重复工作更少。如果员工相信组织能提供支持，他们就会花必要的时间来提高自身能力和技能，加强工作中的联系，减少倦怠风险。创建和维持一个信任度高的环境很重要，这是培养人际关系和理解devops的关键。

信任度高的组织与更高质量的工作也呈现出正相关性。如果一个人来自信任度低的组织，这一点看上去可能不太直观，因为他可能认为反复检查甚至重做另一个人的工作是必要的，这样才能确保不会有遗漏。不过，这种行为，不论是重复工作还是管得太细，都有一个负面效应。一段时间后，如果人们知道最后总会有人重做或反复检查他们的工作，他们就不会对工作投入太大精力。如果他们认为没有得到足够的信任，很难谋取职业发展，他们也不太会有主动性去真正地努力工作。

除了他人的信任，一个信任度高的组织还会对人们的信任本身带来正面的影响。作为更高级的人员，他们的特点之一是能够相信自己的判断，知道什么时候需要寻求帮助，什么时候需要另一双眼睛来审查，以及什么时候他们自己已经具备所有必要的能力。如果一个人不断被其他人质疑，他就无法培养这种能力，不能相信自己的直觉。在一个信任度低的组织中，如果人们被教导要质疑自己的工作，他们就不会挑战自己来发展新技能，而是会墨守成规，只做他们感觉安全的项目，这会在很大程度上限制他们的发展和成长。

如果组织有问责文化或过度竞争的环境，这些组织中通常就会出现信任问题。如果每个人必须只为自己考虑，例如，人们要接受分级评级或者如果犯错就会被辞退，他们就不会相互信任和协作，这将影响沟通和创新。最终，如果人们没有得到应有的信任来完成他们的本职工作，有抱负的人（或者有足够特权可以有更多选择的人）就会离开这里，投奔到他们能得到信任的组织。

## 创新

建立网络使人们可以与组织内以及组织之外的人建立联系，这会促进填补组织中知识或创新的空白。要实现创新性，组织需要有一种信任和无问责文化，人们成功时会受

到奖励，不仅如此，如果适当地处理了失败也要得到奖励。创新需要冒风险，如果冒风险以及由此导致失败会受到惩罚，这种问责文化对于这种存在风险但会带来创新解决方案的创造性行为没有任何帮助。

组织信任与组织协作有很强的相关性，因为会有越来越多的人得到信任，相互联系，被看作是我们自己的群体和圈子中的成员。如果团队、组织和所用的流程中存在信任，这会使公司更能承受风险，因为人们相信并信任他们的同事不仅能处理失败，而且能够从失败中学习。如果太害怕风险和不知道如何处理失败，这意味着，当失败真正发生时将缺乏准备，而没有充分的准备和反应太慢则意味着失败会带来更大的负面影响。

除了信任和协作，创新通常是创造性“跳变”的结果，这不一定遵循一个严格管理的流程。尽管很难具体描述这些创造性流程的特点，不过要知道，创新通常是在最意想不到的情况下产生的，比如正在反复考虑要解决的一个问题时，思路突然跳到另一个问题上。正是因为这个原因，很多人声称他们最大的灵感是在做与当前问题不相关的某件事时闪现的，如洗澡或者出去跑步。类似地，与其他人的交流也可以从意想不到的方向上提供新的想法，特别是组织之外的人。

## 亲密性需求

如果不加努力或者没有适当的条件，将无法建立组织亲密性，这不是一夜之间就能形成的。在这一节中，我们将讨论要建立亲密性需要有哪些条件，有哪些特点。并不是说这里每个方面都必须达到完美，即使在最好的组织里，这也是一个不间断的工作，其重点是要持续不断的提高。不过，如果一个组织在松弛时间、明确的目标和价值观、空间或合作方面存在严重的问题，就会更难体现亲密性带来的所有好处。

## 松弛时间

工作系统中的松弛时间描述了一种未积极工作的状态。通常这种闲散状态会被看作是没有生产力或效率低下，但是在实际中，有意地规划松弛时间对于避免过度分配工作是很有必要的。如果只关注团队和个人贡献的有效性和效率指标，就会更强调可度量的任务，而忽视本质上较难度量的关系性工作。

松弛时间对于处理有变化的工作也很重要，也就是未计划或者比预期完成时间长的工作。为了明确需要多少松弛时间，首先需要了解这个工作的变化性如何，然后在此基础上增加一个缓冲。如果你每周都正好有20小时的计划外工作，就可以规划20小时，另

外再加上考虑到社交网络和个人成长所需的时间量。如果每周有20~30小时的计划外工作，就需要分配更多的松弛时间。

George和General与他们的经理一同来确定这个两人组可以完成的工作量。他们使用看板过程，分配不同种类的任务作为计划外工作，一个月之后，他们检查了 workflow 指标，认识到每周都有20小时的计划外工作。综合考虑社交时间、个人学习和计划外工作，他们决定分配每周40小时的松弛时间，留40小时作为可以调度的工作时间。



一般地，一个给定角色的工作越容易被中断，就越需要更多地分配松弛时间。因此，与其他团队（如开发团队）相比，运维团队往往需要更多的松弛时间，从而能有效地处理一些意外情况带来的计划外工作，如意外断电、为基础设施应用紧急补丁以及处理其他团队的请求。一个团队工作负荷越不可预料，需要的松弛时间就越多。另外松弛时间对于工作-生活的平衡和个人健康也很重要，这是有效的创造性思维的必要条件。

## 明确的价值观和目标

很多团队发现，如果他们没有明确地对某件事情做好计划，这件事就无法完成。当前，由于很多工程环境节奏很快而且压力很大，很难很好地满足所有既定目标，更不用说目标不明确的情况。要为发展职业关系做出规划，但很多情况下这并没有得到足够的重视，而如果没有相应的规划，这就不太可能发生，导致团队无法得到前面描述的亲密性可能带来的诸多好处。

个人贡献者和他们的管理者需要理解关系性工作也是他们的工作中很有价值而且很重要的一部分。如果给他们安排的工作过多，可能就会认为与其他人联系和“浪费”时间是没有意义的。要认识到工作有多种形式，这至关重要，即使一个人没有坐在办公桌前积极处理客户支持ticket，并不意味着他没有对团队做出有意义的贡献。

在老一辈系统管理员或者在大型公司里工作很长时间的人们中通常会看到类似这样的态度，因为多年来他们所了解的工作就应该是怎样。不过，正如我们指出的，建立和维护关系会对个人和团队带来很多好处。需要明确地重视这个工作，让大家清楚地了解期望。管理者和团队领导应当指出所期望的结果，而且要向完成目标的个人表示祝贺。



可以用技能矩阵来定义一个组织或角色中不同层次所期望的技能和能力，这对于为这种行为明确地设定目标会很有用。这个矩阵中可能有一些行强调技术或“硬”技能以及所完成的项目，不过，还应当有一些行强调人际交往和协作能力。要重视谦逊和善于倾听的特点，这对于团队和组织很重要，通过为工作伙



伴提供时间和空间来分享他们的观点，并且认识到没有哪一个人最重要或者能提出所有解决方案，从而促进关系的加强。

如果按照类似这样的一个体系，有助于确保人们得到公正的评价，可以考虑到他们所做的所有不同类型的工作。明确地重视建立关系还可以帮助避免这样一种情况：某个工程师可能水平很高，能力超群，但是人际交往能力很差，如果整个组织都知道这种行为不可忍受或者不会得到奖励，就能避免这种情况发生。度量和奖励什么，通常就会得到什么，“硬”技能和“软”技能都是如此。

## 空间

在公司里，要建立一个大家共享的休息空间，利用适当的基础设施允许不同团队建立协作，激发创造性并促进问题解决。这些共享空间不应以个人的工作空间为代价，因为这是对别人的不尊重。室内咖啡厅是一个很好的选择，可以为当前的任务提供一个让精神放松的场所，也可以让身体放松。不过要记住，并不是所有人都喝咖啡，所以要提供含咖啡因和不含咖啡因的多种不同的饮料，这是确保每一个人都感觉到他们参与其中的关键。

有些人可能担心离开办公桌和“真正的工作”时间太久。前面已经提到，关系性工作的价值往往被低估，特别是没有看到这种工作还有一个重要角色，它们是“真正的工作”的一个催化剂。建立这些小空间可以鼓励个人和团队更自发地收集他们需要的信息。不过，还要考虑到这些空间要尽可能做到不偏向任何一个团队（例如，不要把所有可预约的会议室都安排在某个团队的办公区）。遵循这个原则，将促进团队间以及团队内关系的建立。



要有多种不同大小和配置的空间支持人们协作，这很关键，尽管有些组织可能因可用的办公空间所限，而且不是每一个人都有能力或资源随意改造他们的空间。不过，如果有一些房间，从单人的电话间（这样与同事或开发商打电话时就不会影响到旁边的其他人）到10人的会议室，从2人间（这样两个人可以结对工作）到开放空间，人们可以随意地自由来去，这就允许人们根据给定时间他们所要完成的工作来选择适合他们的协作方式。更进一步，为人们提供时间和空间来加强联系还可以降低流失率。我们常常会因为周围的人的原因在一个职位上待更长时间。

对于开放办公计划还存在一些问题，其中一个问题是要想完成某个工作而不让旁边完成其他工作的人分心，往往会更难，这包括协作或结对工作。一个工作有意义，并不

意味着它能安静地完成，这个工作有可能很吵，会影响旁边想努力解决某个棘手问题的人。如果你有一个开放办公计划，但是没有隔离的空间能避免人们相互影响，关系或生产力（或二者都有）就会受影响，所以在评估办公空间租约或做出改变时一定要记住这一点。

## 协作和合作

除了提供空间和鼓励让员工相互合作和协作外，如果想要得到所有相关的好处，组织还需要有一种文化，能真正促进和奖励关系的建立。这意味着要奖励涉及或鼓励合作的行为，同时要避免有意或无意地培养一种竞争环境。

在第二部分我们提到过，使用一个诸如分级评级的系统对员工绩效进行评级时，这就与合作环境的主旨背道而驰，这是因为，为了让某个人在分级评级游戏中“胜出”，必须确保周围的人“落败”。如果让人们相互之间为了晋升、加薪甚至只是为了保住工作而竞争，几乎可以肯定，在这个组织中不会看到真正的协作。

类似地，要当心人们或系统奖励那种间接告诉第三方的行为，而不是与相关的人直接解决问题。消极对抗或其他间接行为会削弱信任，而这对于合作至关重要。不过，有一点很重要，如果有些人在工作场所感觉难堪或者不安全，这就不适用：很多情况下，人们可能认为直接指出这种行为不太安全，而向他们的经理或HR提出他们的顾虑感觉会更好一些。

基于所有这些因素，可以建立一个非常有效的协作和亲密性环境。

## 度量亲密性

亲密性很难度量。你可以度量亲密性的结果，而不是亲密性本身。但这不影响主动地鼓励发展社区。另外，可以注意一些信号，包括组织中的成员相互之间以及团队之间在发展关系，以及发展与组织之外的联系。

## 员工技能和评价

要设定明确的目标，并清晰地定义亲密性和协作的价值，这是实现这些目标关键而必要的步骤，在员工反馈机制中定义这些目标也是其中的一部分。一个好的技能矩阵或审查工具会把重点放在个人沟通能力等方面。

显然，这个领域更强调沟通质量而不是沟通数量，在这里同伴反馈可能很有用。例如，考虑这样一种情况，员工相互之间需要信息或辅助。人们知道要向谁提问吗？

他们愿意去问合适的人吗（即使那些人不在他们的团队里）？他们会不会不愿意寻求帮助，即使拒绝得到帮助意味着自己的进度会减慢或者甚至会妨碍到其他人？他们是不是总是因为一些争论而使会议、代码审查或电子邮件线索脱离正常轨道（尽管大多数人认为这些争论没有必要或者会起反作用）？

另一方面，要注意这样一些人，他们被看作是很好的信息源，也就是人们需要帮助时总会找的那些人。最好的沟通者就被看作是这样的信息源，这通常是因为他们有妥善的响应方式：他们不会表现出优越感，不会小看向他们提出的问题，会帮助身边的人获得答案，即使他们自己并不知道答案，或者在忙于其他工作，也会积极给予帮助。好的协作者会真心希望帮助身边的其他人提高和成功。

根据你关心的特点建立技能矩阵、晋升流程和评级系统，你就能看到人们在协作方面存在哪些优缺点。直接从员工的同伴得到这个反馈，这可以是他们的直接团队伙伴，或者是其他团队里与他共事的人，这样就能掌握他们的日常工作习惯，这些信息原本是管理者们没有机会看到的。

## 团队间交流

如果你的组织在完成某种工作或项目跟踪，涉及多个团队时，这对于度量工作的进行状况可以提供一个好的起点。如果客户ticket在多个不同的支持人员之间流转，所有人都很努力地更新ticket，这就是一个很有价值的信息源，由此可以了解工作的完成情况，以及涉及到哪些人。如果你认为做不到这一点，因为人们根本不更新他们的ticket，这将作为另一个可以度量和跟踪的因素。

有些工作很小，只需要单独的ticket就可以解决，相比之下，取决于你的组织用什么方式跟踪较大的项目，这可能是间接度量团队间亲密性的一个好方法。应当指出，尽管不是严格要求，但是如果每个团队都使用共同的软件来跟踪和组织他们的工作，而不是每个团队都使用各自不同的系统，工作会容易得多。这里可能关注的方面包括：

- 一个给定项目涉及多少个团队？
- 团队之间的工作如何细分？工作划分是否均衡？这些划分是否合理，或者是否有一个团队或团队成员相对其他人负担过重？
- 项目生命周期的不同阶段需要多少时间？具体地，规划需要多少时间，这个阶段的团队组成是怎样的？



- 团队或项目成员之间是否经常出现误解？这些误解特定于一个特定的团体还是与沟通方法有关？

在特定的跨团队项目以及他们的常规工作中，有一点很有意思：人们是否会经常寻求其他团队成员的帮助。人们是否愿意寻求其他团队成员的建议？是否经常希望得到解释和说明，而不是想当然地假设自己知道其他人想表达什么？是否经常找其他人结对工作？通过这些方面也能很好地查看各个团队，不过如果你注意到这些行为只是经常发生在一个团队中，而不是在所有团队之间，这可能是一个信号，说明某个方面可能存在问题。

可以查看多少个项目由多个团队完成，另外多少个项目仅限于某一个团队的成员，这也很有好处。如果工作需要多个团队的参与，这种参与是同时发生还是一个团队先完成他们的部分，然后把他们完成的结果“转交”给另一个团队？尽管这种转交不一定是坏事，但是这可能是一个信号，表示整个过程中可以有更多协作，应当进一步调查来看是否可以做某些改进。类似地，一个项目仅由一个团队完成可能不算问题，但是如果这个工作的结果将由其他团队使用或者会影响其他团队，就要看看是否有办法在整个过程中纳入那些团队。

## 寻求社区支持

devops运动之所以如此生机勃勃，很重要的一部分就是它的社区，从业人员充满热情，他们很愿意聚集在一起讨论他们如何工作以及在做什么。与数十年前不同，现在行业技术和实践不再笼罩在一种神秘的气氛中。实际上，这个领域最知名的一些公司多年来一直在公开介绍他们的经验，不仅包括他们的成功，也包括他们的努力和失败。

Etsy把这个想法称为“慷慨的精神”，这家公司的工程博客Code as Craft以及大量开源工具（包括StatsD）让它颇负盛名。这种慷慨包括撰写相关工作的博客文章，在行业大会上发言，或者为开源贡献力量，而且鼓励员工至少每年参与其中（或更多）活动，以确保不断地丰富和回馈社区。任何组织如果从会议发言、博客文章、会谈或开源项目中受益，都应当按协定努力做出回馈。

正是因为人们愿意自由、开放地分享他们的工作和想法，使得这个社区不仅很强大而且很有价值。人们通过Twitter、LinkedIn或各种会谈和大会等媒介沟通想法时，就会建立原本无法建立的联系和纽带，发现自己想不到的解决方案和新想法，而且不会浪费时间去处理已经得到解决的问题（这要归功于不断发展的开源软件体系），使我们能更好地利用时间和精力。

在这一章前面讨论过，要成功地建立合作的社区，很重要的一部分是需要秉承这样一种思想：坏份子要受到惩罚，这是指以团体的利益为代价突出其自我利益的人。只从社区攫取营养而从不回馈不是一种合作行为，尽管其他组织由于其慷慨的精神很可能会继续向社区分享他们的知识和想法，但是倘若他们不再分享，就将是行业的损失。

## Sparkle公司开发和运维的亲密性

“我很高兴可以减少在这个评论特性等用例上的开发时间，让我们能有更多的时间提高服务的最终用户体验。根据当前的计划，我想如果你们认为值得，我们可以对这个评价特性做最后冲刺，开发一个原型”，Hedwig在演示结束时说。

“我对MongoDB的经验不多。我需要再和团队的其他人讨论一下，看看他们有多少经验，另外接手一个新项目的难度如何”，George很谨慎。“作为Sparkle公司的新运维工程师，我不希望运维团队的工作不可持续”。

“接下来继续对这项工作进行评估，不断更新沟通情况。George，请让Geordie、Josie和Alice加入这个项目。我会与运维团队领导协调，确保一旦我们得到更多信息，这两个团队能够在决策时发表意见”，General说。

## 小结

发展和维持开放、信任和沟通的关系在个人之间以及一起工作的团体之间都同样重要。我们认为自己是哪些团体的成员，这会对我们自我描述的身份有很大影响，根据我们对团体从属关系的认知，这还会进一步影响我们如何与人们交流和共事。

要建立协作性的组织和行业，关键是要寻求方法破除团队间的壁垒，扩展我们的团队从属关系和定义，从而可以在个人、团队甚至公司之间让工作、信息和想法更自由地发展和流动。可以在一个组织的不同团队之间以及甚至在不同组织之间分享故事和想法，这会带来更多信任和更多创新，而且有助于维持相互理解，这对于建立devops环境至关重要。

## 亲密性：误区和问题排查

关于亲密性的误区和问题与个人协作和沟通的有关问题类似，不过在一个更高的组织层次上。

### 亲密性误区

对于一个组织中不同团队的职责和贡献，人们通常有不同的想法，另外对于devops环境中亲密性和分享的意义也往往有不同的理解。

### 运维工程师不如开发人员对组织有价值

尽管通常并不特指运维和开发团队，不过人们常常认为有些团队原本就比其他团队对公司更有价值，这种想法可能很难撼动。之所以会这样认为，部分原因来自于有形和无形工作的区别。对于不一定理解团队日常工作细节的人来说，会认为最后在客户面前展示的产品的相关开发工作更实在，比如设计团队集成和展示的模拟原型。而有些工作只有在有缺陷或完成得不好时才会被看到（比如可以想想看网站服务中断，或者一个态度恶劣的客户支持代理），对于这些工作这一点尤其突出，与正面的事件相比，负面事件会更清晰地一直留在我们脑海里。

组织的亲密性和团队间的关系之所以有巨大的威力，就在于团队可以相互支持和帮助，而不是彼此阻碍。一个运维或内部工具团队可能会妨碍开发人员部署代码，也可能帮助开发人员轻松地建立开发所需的测试环境。如果说开发人员在帮助客户，那么能帮助开发人员的人就是在确保客户得到更多的帮助，这样一来，开发人员可以花更



多时间开发和修正客户最后面对的产品，而不用等待漫长而且问题多多的部署过程，也不会在尝试测试所做的修改时没有一个专业的测试或开发环境。

如果有指派运维人员之类的计划，或者只是鼓励人们（包括个人和经理）参加其他团队的会议，就能使组织中不同部门完成的的工作更有可见性。这有助于消除一些误区，如认为某些团队没有做多少工作或者没有做有意义的工作。当然，可能某个团队确实没有完成它本可以做的的工作，但是更多的情况下，这只是一个组织契合的问题，而不是团队或其成员的固有价值问题。

尤其当组织发展和变化时，不同团队或产品可能没有原先那么有意义。要关注不同的团队及其工作如何在整个组织的上下文中整合在一起，另外还要注意，对于什么是工作，看法不要太过狭隘。

## 在公司外分享太多会削弱我们的竞争地位

如今确实处处有竞争，一个公司不希望做任何可能影响其行业竞争优势的事情，这是可以理解的。这种观点通常会导致公司禁止员工做在大会上发言或开发开源软件之类的事情。在这里存在一种担心，认为开发开源软件就是提供免费的东西，而没有通过销售来谋利，而大会发言可能会为竞争者提供灵感，帮助他们取得成功（而且员工参加大会还会占时间，而不能去做“真正的工作”，这在前面已经解释过）。

不过，最终看来，大多数人考虑在大会上展示或者在devops社区里提供开源支持的工具和技术与为他们公司盈利的工具和技术并不一样。例如，Target是一家零售商，靠向顾客销售实体产品来挣钱。如果他们发表一个大会演讲，介绍如何开发软件来增强顾客面对的Target网站，这并不会交出他们销售的产品，而且根本不会影响到产品。他们的员工在devopsdays会议和其他行业会议上发言时，所讨论的是他们如何支持和创建，这让他们在这个领域很有竞争力，而不是分享竞争秘诀本身。

这是一个由社区发起的运动，人们在一起讨论他们面对的问题，并尝试找出这些问题的解决方案。开发人员和系统管理员希望讨论他们面对的文化和技术挑战，而这些挑战通常都不是领域特定的问题。如今常常说很难找到好的技术人才，如果限制你的员工参加专业社区，这会影响你工作招聘的吸引力，而这将是真正的竞争劣势。

## 亲密性问题排查

亲密性方面的问题排查通常也是间接的。要在你的组织中建立和维持一种开放、协作的文化，这个过程可能出现一些问题。在这一节中，我们将给出一些提示，指出如何识别和解决其中的一些常见问题。

## 一个或多个人中断团队涌流

上一章提到过，团队涌流在本质上有别于个人涌流。一些个人可能通过身体或语言上的强势行为破坏团队涌流。有时这些人被看作是组织中的关键人物。如果一个组织没有直接解决这些行为，就是在暗暗支持和强化这些行为。

除了影响团队涌流，破坏性行为还可能带来压力和困扰，导致其他团队成员离开这个组织。破坏性行为的例子包括轻视、翻白眼、单独回避、拒绝指导、拒绝帮助其他人和扔东西。要了解如何响应这些行为，首先要理解为什么这个人会做出破坏性行为。其原因包括权力变化、转嫁挫折以及冲突。

高效的组织会认识到团队和协作的重要性。第一步是建立一个承诺，明确阐明后果来消除这些行为。通过教育，并为个人建立一个安全的空间对违规行为进行沟通，确保整个组织开展协作。员工可能会担心遭到报复，组织应该有相应的合同条款来减少这种恐惧。要建立一个行为守则，定义哪些行为可接受，哪些不可接受，并明确管理这些行为的过程。Geek Feminism网站提供了建立一个有效的行为守则的指南。

如果违反了守则，重点应该放在违规行为上，而不是人。人们不一定总能了解他们的言语或行为产生的影响。个人需要理解他们的行为与这种行为对其他人的影响之间存在的直接关联。如果行为没有改变，再次违反行为守则，就需要另外采取措施。取决于公司的具体情况，可以有多种不同的选择。例如，管理愤怒情绪的培训班或者指导培训都可能很合适。如果这个问题是由于工作环境中长期不断的压力所致，要找出压力过大的那个人，让他有足够的休假。如果问题是由于冲突造成的，就需要采取措施来解决冲突，如果需要还可以进行调解。

如果补救措施没有帮助，也有多种选择，可以重新把这个人安排到另一个团队，或者为他寻找不要求团队工作的机会，也可以干脆让这个人离开。辞退一个员工并不一定是负面的。应当指出，对于看起来不能与其他人很好合作的人，如果为他开太多特例，这会成为一个不好的先例，可能导致更多的人认为这种不好的行为是可以接受的。

## 一个团队不断妨碍其他团队

如果你发现一个团队或团体看起来总是在阻碍其他团队完成他们自己的工作，首先要做的就是调查哪些因素会造成这种阻碍。如果工作不能足够快地完成，这个问题可能说明负责这项工作的团队配备的人员不足，无法承受为他们分配的工作量。考虑本章前面介绍的指派运维工程师的例子：根据运维团队的规模，同时考虑到他们要支持多

少个其他团队，以及那些团队的工作负荷，就可能存在这种情况，没有足够的人员、时间或适当的综合技能来做到要求他们完成的每一项任务。

如果团队相互之间不能充分理解对方的问题、项目和需求，这也是带来阻碍的另一个常见原因。特别是当不同的团队有完全不同的目标、优先级和需要满足的KPI时，对一个团队极其重要的工作在另一个团队看来却有可能根本不合理，这就会导致这项工作不被重视。这可能是由于缺乏沟通造成的。如果只是通过重新分配ticket之类的形式来回传递工作，接到工作的人可能没有足够的上下文来充分理解它的重要性，所以要确保在工作转手时相关的人有足够的沟通，这会很有帮助。

即使进行了沟通，仍有可能存在误解。所涉及的两个团队应当确保在工作交接的最初就要对这项工作有尽可能清晰的理解，包括需要做什么工作、最后期限、有哪些其他需求、为什么这项工作很重要以及它的优先级是什么。越早澄清误解，产生的阻碍和延迟就会越少。

可能存在这样一种情况：由于公司的政策、兼容性问题、技术限制和其他一些原因，导致所请求的工作无法完成，而且取决于组织环境，这些原因不一定总是很明显。有些人（或者在某些文化中）发现很难直接拒绝别人的请求，或者感觉这样过于粗鲁，所以要注意文化差异以及非语言交流，这会很有帮助。

组织环境中的其他一些因素有可能在培养一种竞争的环境，而不是一个协作或合作的气氛。如果两个团队直接竞争某些资源，如预算或人员，特别是如果这些团队有不同的目标，要达到不同的指标，那么很遗憾，他们不会有相互帮助的初衷。如果确实是这种情况，可能有些问题必须在组织层次上解决，而无法在个人或团队层次解决。

针对造成阻碍的很多原因，直接沟通可能对解决这些问题很有帮助，但是有一点很重要，要带着正确的态度加入这些讨论。如果看上去有人有意与你作对，很可能会认为他们是恶意的或者能力不足，这种假设会让你戴上有色眼镜，并在语言和行动上表现出来。另一方面，如果人们感觉他们总是被质疑，或者他们的工作没有得到认可，就会有防卫心理。这两者都将成为滋生不良表现和消极对抗行为的土壤。为了达成一致，双方必须记住大家的契约，即每一个人都在为同一家公司和相同的总体目标而工作，而且同意公开、直接地沟通，对当前状况和每个人的期望重新做出评价。

## 一些团队感觉不被重视

本章前面提到过，在技术公司里有这样一种趋势：往往会过于强调开发人员，而忽视其他不那么“光鲜亮丽”或者不太热门的团队和角色。可以理解，这可能导致这些团



队存在一种不满的情绪，认为相对于他们在行业中的价值或者相对于公司的其他部门，他们没有得到应有的重视。尽管开发人员确实很重要，但是要建立一个成功而且可持续的企业，不光是要能写软件，还要包括很多其他方面。

当然会有一些不可控的经济因素可能对薪水等方面有影响，不过通常公司确实会左右一些方面，如整个公司如何发放补贴、补助和奖励。开发人员和其他工程岗位的人通常会得到参加大会、会议行程和类似活动的预算，这对于发展他们的专业技能和职业社交网络很有好处。要确保不只是工程部门可以得到这些机会。

如果你的公司主办会议时会给工程师发放精美的铭牌，其他部门也可能会感觉被忽视。非工程师也希望能够展示企业精神，尽管一两件T恤或卫衣可能看起来不是大问题，但长期来看这确实能大大增强士气。另外不用多说，公司发的T恤和卫衣从一开始就应当量体裁衣，如果公司里有些女性没有得到适合她的衣服，她肯定不会说这个公司的好话。

前面我们讨论过建立空间来发展亲密性文化的重要性，这个领域也经常会看到不平等的现象。如果一个人整天坐在办公桌前面，不论是在写代码还是在回复支持邮件，都会感觉不舒服。不要只是为工程师预订更好的座椅、提供更舒服的办公空间或者精美的人体工学立式办公桌。另外还要注意是否所有人都能使用会议室和其他协作工作场所，确保所有员工都有同样的机会，而不会优先考虑某些员工。

最后，注意哪些团队和员工最经常受到表彰，可能在公司博客或员工页面上有他们的宣传页，或者他们完成某个项目时会在一个公共休息区开庆功会，或者是在全体参加的季度大会上得到表扬。对很多人来说，都希望他们作出的努力和成就得到认可，这是工作满意度中很重要的一部分，而且如果得到赏识和奖励，即使奖励很小，也会让他们更能接受原本艰巨而漫长的工作。



我们不建议在招聘启示中把开发人员称为“明星”，同样重要的是，不要在办公室把他们当作明星，而把其他团队和组织只当作后台工作人员。这只会滋生不满的情绪而不是促进协作。

## 看上去人们相互不信任

鼓励、发展和维持信任不是容易的事情，尤其是在一个之前没有信任的环境中。信任，不论是对同伴的信任，还是对管理者和领导人的信任，都很难得到，这不是想要

就有可以随意挥霍的东西。如果你的环境中看起来缺乏信任，几乎可以肯定，这个信号说明存在着需要解决的文化问题。

问责文化不会带来信任，因为人们会由于错误而受到惩罚，而不是简单地承担责任和学习。我们在第一部分和第二部分中简单谈到问责和无问责的概念。如果你还没有读过这些章节，现在要处理组织信任问题，可以把这些章节作为一个不错的起点。如果你的组织刚开始从一个问责文化转换到无问责文化，要理解文化的转换需要一定的时间。要重建失去的信任，与从一张白纸开始建立信任相比，将需要更长的时间，所以如果正在从一种倾向于问责的文化开始恢复，一定要理解这种事情不是一蹴而就的。

整个组织范围内开放的沟通是在组织中建立信任的关键。如果管理者和领导把门关上（可能是真正的门，也可能是隐喻的门），在门后面暗箱操作，那么只要求个人贡献者保证开放和诚实是不够的。如果只是把门打开，这也不够。大多数时候，人们会把这看作是一种空洞的形式，或者不能或不愿冒险站出来问一些难堪的问题而让自己受人瞩目。即使在最好的组织里，关于补偿等方面也经常存在一些问题，出于很多原因，包括个人原因和文化原因，人们不愿意直接询问。通过定期举办研讨会，员工可以匿名提交和查看问题并投票，然后让HR、经理或主管领导回答最普遍的问题，这会带来透明度，并从上到下建立起信任。

有一点很重要，如果你要这样做，就要100%做到。尤其是在当今的世界里，人们和公司总是试图向我们推销某种东西，人们很清楚什么情况下没有听到事实，有可能是彻彻底底的谎言，也可能没有得到任何回答，他们的问题只是淹没在公司的层层推诿中。大多数人更愿意听到“我们不能（或不想）回答这个问题”，而不是得到一个没有任何实质内容的回答。

简单地说，信任和沟通需要以身作则，这可能意味着要确保公司所有层次的管理者和团队领导人要经过充分的培训，使他们在建立信任、开放地沟通以及处理冲突方面能够做到游刃有余。

一旦文化能支持和维持信任，就要让人们开始交流，可以先从压力和风险相对低的方面入手，这是在人们和团队之间建立联系的最好的方法之一。可以尝试“搅拌器”项目之类的方法，选择加入的人可以与其他团队的个人随机结对，然后鼓励他们交谈一个小时左右，彼此了解对方，这通常是在午餐或喝咖啡的时候。这会是扩展人们联系的一种很好的方法，而且没有必须共同完成具体工作所带来的压力。一旦人们熟悉了这种交互，再为这些两人组或小组随机分配一些任务，如处理低优先级的bug日志，这也非常有助于建立信任和亲密性。



## 人们只关心工作的技术方面，而不关心人际问题

要强调亲密性、协作和合作，对于这种想法，我们最常听到的一种反驳是：这会让人们太过分心，不能很好地完成本职工作。一个组织的主要目标不是要建立友谊或其他人际关系，这一点不假，但是如果忽视人际关系对组织的影响，不论影响是好是坏，都是目光短浅的。持有这种观点的人对于什么才是“真正的”工作抱有一种很主观的成见，但是这忽略了一个事实：对于完成工作的人，影响人们生产力的任何方面（个人层次或团队层次）都会对“工作”产生显著的影响。

没有人或团队能够在真空中存在。即使在最小的创业公司，员工与客户之间、员工与预期的投资者之间、员工与预期的员工之间，都会存在关系。一个组织越大、越复杂，这些关系对于完成什么工作以及工作如何完成的影响也越大。可以考虑一个老式企业，完成工作往往需要经过多层领导审批和多道手续流程。这些流程正是公司里不同的人、团队甚至组织之间众多不断发展变化的关系带来的结果。通过检查一个组织中的关系，可以帮助你确定存在哪些痛点，而且努力发展上述关系可以逐步地缓解痛点问题。

复杂的组织有很多不同的团队，它们会分别开展工作，至少有某种程度的独立性，这里可能存在冲突。如果没有一种方法识别和解决这些冲突，这些不同的目标和优先事项将无法保证组织的整体成功。投入时间和精力来建立关系和培养能力可以鼓励团队和个人合作而不是竞争，另外可以帮助组织完成它的集体工作。所有人都曾经历过某种过于官僚的流程并深受其害（例如，George必须向他的经理提出请求，这个经理又要与General的经理讨论George需要General做的工作，而不是与General自己直接讨论），工作中不合适的关系绝对会妨碍完成“真正的”工作的进度。

团队动力也会对团队士气产生影响，这会进一步影响生产力（我们在这一部分和第二部分已经做过讨论）。所以尽管工作不是要交朋友，但是应该把我们的眼光放远一些，更全面地考虑什么是工作，这很重要。不只是坐在办公桌前写文件或写代码是工作，建立关系使组织作为一个整体进行协作并有效地运行同样也是工作。

## 不同团队看起来根本无法一起工作

在成熟组织中，往往已经有一套不易更改的关系和习惯，让人们改变根深蒂固的观念可能很困难。如果团队和团体习惯于相互竞争，那么没有人想做第一个吃螃蟹的人来改变这种行为；否则就像他们表示屈服或者承认自己“输了”。如果人们习惯于这种场合，改变行为就意味着削弱他们的地位，因此他们会强烈地反对做出改变。



不过，如果团队一直以来都在相互竞争资源，有着完全不同的目标，或者彼此之间严重割据，倘若不对组织文化或周围环境做出改变，就不能指望团队改变他们的行为。从以某个人离职或降职而告终的事后分析会转换到无问责的事后分析，重点应当是学习而不是惩罚，可以优化团队用来分享或沟通的流程或工具，或者甚至可以重新组织团队，这只是几个例子，要让个人和团队开始改变原来的行为，还可以采取很多其他做法。

信任是团队合作的一个必要要素，建立信任不会是一蹴而就的，需要一种适合的文化来培养信任。要检查所在环境，查看团队和组织的压力和行为，了解团队是否存在信任问题。

## 以前的人际冲突导致现在的团队间冲突

在一个决定完成某种“devops变革”的组织中，经常会出现这样的场景：有些团队出于历史上的原因相互之间一直有冲突。最通常的情况是，这种摩擦往往发生在开发和运维团队之间，因为这两个团队的目标在很多方面都不一致，不过任何要一起工作但是目标和动机不同的两个团队之间都有可能发生冲突。

你会看到，即使已经在组织层次上想办法来保证目标更一致、重新分配资源或调整流程，或者尽可能减少团队之间的摩擦和冲突，但是这些团队中的个人还是有可能无法合作。我们都是普通人，比如，工程师可能想象自己极其理性，但是有些情况下还是会情绪化，特别是以前的冲突变得相当白热化时，我们的情绪就有可能失控。

如果团队之间的冲突看起来是普遍存在的，没有明确的来源，如果有可能，可以在团队或项目之间重新分配人员，这通常会很有帮助。人们可能已经形成一些小集团，这会成为滋生抱怨和对立心态的土壤，而重新分配人员有助于解散这些小集团。人们在与其他人的交互中还会形成一些习惯。他们可能会打听关于“开发团队”或“运维团队”的信息，这会触发某种本能反应。重新分配（或者甚至重新命名）团队有助于打破这些老习惯。

有些人之间可能存在一些不可调和的冲突，这会影响所在团队其他人的行为。如果更高级的人员（在一些组织中，这些人往往也是更有影响力的人）存在负面情绪而未能解决，这种情况尤其突出。与管理者、指导者甚至同伴定期的一对一交流有助于找出这些冲突，有时让人们坐下来谈一谈会很有好处。如果有足够的时间，而且团队或组织中其他方面有积极的变化，你会发现，人们会把自己的想法讲出来，会对以往（真实或假想）的过失道歉，开始修复他们的关系。

## 某团队看起来想成为一个孤岛

与上一种情况中讨论的小集团类似，可能还会有这样一些团体，他们与所在团队、部门甚至组织中的其他部分完全隔离。看起来尽管大多数人都对devops变革带来的改变持开放态度，比如重组的团队、新工具或改变的工作流程，但是一小部分人会对这些改变非常抗拒。

你会发现，以前没有得到认可或者未能受到充分赏识的人往往存在这种情况，如IT技术人员甚至运维工程师。这种心态会导致人们把信息私藏起来，来保护自己的职位安全。由于没有适当的方式让他们感受到工作被认可，他们就会找一种方法使自己在其他方面得到认可。

devops运动更加强调之前数年或数十年没有得到足够重视或尊重的角色（如运维等角色）的重要性，不过即使是开展了类似这样的一些运动，有些领域里人们仍然没有感受到被认可和尊重，在工作场所中还是没有安全感。devops在理论和实践中有时会在分歧，在这些领域，你会看到一些团体的工作没有得到应有的认可，这些团体被同伴“小看”，或者人们总是以一种问责（而不是无问责）的态度对待他们。或者，他们也可能根据以往的经验认为会有这些一些行为，因此自我封闭起来，而不会根据事实做出调整，这也是对以往问题的一个本能反应。

解决类似这种问题时，通常需要明确这些员工的哪些需要没有得到满足。马斯洛的需求层次理论会很有用。对于基本需要，要考虑公平的补偿。在安全方面，员工需要确信他们的职位不会丢。组织能重视他们的工作，而且能提前告知可能带来影响的情况，如可能的裁员。在归属方面，人们需要感觉工作环境是舒心的，能得到管理者和同伴的赏识，所以要注意那些不尊重别人或者不提供有价值反馈的人。最后，自尊和自我实现意味着员工对自己和所做的工作感到自豪；如果某些职位只是作为后台工作，人们对这些工作和组织的感觉肯定会降低。

要注意其中某个需求（或所有需求）是否未能得到满足，这有助于指出需要在哪些方面改善与那些看起来很孤立的团体或团队的关系。与本书中讨论的其他关系一样，这些工作关系建立在信任之上，要建立、维持或修复这些关系需要投入一定的时间和努力。当然，可能有些人总是站在对立面，拒绝修复信任，他们无法适应这个不断改变的组织，这些人可能不适合留在你的组织里。

## 人们在批评devops的错误

做出重大的改变肯定少不了遇到困难，相对来讲，有些人总是更加抗拒改变。一个组



织在转换阶段中，往往会遇到这样一种情况：对于正在进行的改变，持反对态度的人（不论是出于什么原因）总会因为某些问题或错误提出批评。一个组织开始转向 devops 文化时，可能会有一些人非常抵触，极力反对这些改变。

例如，假设一个组织正在努力从一种频率很低的手工部署过程过渡到一种更自动化的持续交付过程。一开始，新的自动化部署工具可能并不完美。因为与所有软件一样，总是会发现需要解决的 bug。抗拒改变的人就会指责这些新工具带来的新问题，指责 devops 本身，甚至指责支持这些改变的人。他们可能说，“原先本来一切都很好”或者“如果我们照老办法，就不会遇到这些问题”。他们认为问题在于 devops 本身，而没有意识到每个新工具或新流程在发展过程中都会有问题，需要时间来适应。

有一点很重要，如果管理层想要成功，就要自上而下地对 devops 新项目提供支持。如果组织的领导人因为少数大声抱怨的人而摇摆不定，就很难有持久的改变。改变不会一蹴而就，在这个过程中出现错误或 bug 是正常的，这也是可以想见的。由于没有一种十全十美的解决方案，所以需要做一些尝试，经历一些错误，才能找出对你的组织最适用的工具和流程。

要提供适当的途径，使人们能够提交反馈，了解改变对他们有什么影响，以及他们如何看待正在进行的改变，不过要注意出现负面反馈的频率和这些负面反馈的来源。如果很多人都发现某个解决方案对他们不适用，就肯定需要进行调查和做出调整，不过，如果只是有少数负面的声音，产生很大的噪音，一定不要让他们占上风，毕竟改变对大多数人都有益。不是所有人都完全适合每一个组织，有些人总是对改变持反对态度（或者特别反对 devops），对于一个发展的组织，这些人可能并不适合。



# 工具：生态系统 **工具**

第4章中关于增进团队之间的理解介绍了一些定义和术语。开始讨论如何使用工具来建设和维护文化的各个方面之前，我们将在第4章的承诺之上再做一些扩展。这里肯定无法包括有关的全部技术或术语。

对于这些术语和概念，人们可能有不同的认知模式或不同的理解。通过明确共同的含义，就能更细致地开展讨论，并且更好地理解这些概念。

## 软件开发

软件开发工具可以帮助完成应用和服务的编程、建立文档、测试和修正bug等过程。这些工具并不仅限于特定的角色，对于能某种程度上参与软件的任何人都很重要。

## 本地开发环境

要让员工很快就能对产品做出贡献，拥有一致的本地开发环境是至关重要的。并不是说人们只能使用某个标准编辑器，没有任何灵活性或者不能有任何定制，而是说要确保员工能得到所需的工具从而有效地完成他们的工作。

在你的环境中，根据个人喜好会有不同的配置需要。有些人需要多个显示器或显示器，或者需要高分辨率显示屏让人们能看清细小的代码变化。有些人需要定制鼠标、鼠标和其他输入设备。要让你的本地开发环境成为团队标准，让配置变得简单。你和团队之间是否有一致的框架。如果有一致性，那么，新员工能很快上手，并能很快做出贡献。

# 工具：生态系统概览

第4章中关于增进团队之间的理解介绍了一些定义和术语，开始讨论如何使用工具改进和维护文化的各个方面之前，我们将在第4章的术语之上再做一些扩展。这里肯定无法包括有关的全部技术或术语。

对于这些术语和概念，人们可能有不同的认知模式或不同的理解。通过明确共同的含义，就能更细致地开展讨论，并且更好地理解这些概念。

## 软件开发

软件开发工具可以帮助完成应用和服务的编程、建立文档、测试和修正bug等过程。这些工具并不仅限于特定的角色，对于在某种程度上参与软件的任何人都很重要。

## 本地开发环境

要让员工很快就能对产品做出贡献，拥有一致的本地开发环境是至关重要的。并不是说人们只能使用某个标准编辑器，没有任何灵活性或者不能有任何定制，而是说要确保员工能得到所需的工具从而有效地完成他们的工作。

在你的环境中，根据个人喜好会有不同的最低需要，可能需要多个显示屏以增强协作，或者需要高分辨率显示屏让人们能更舒服地长时间查看，可能还需要特定的键盘、鼠标和其他输入设备。要让你的本地开发环境满足当前标准，这包括确定团队内部和团队之间是否有一致的框架。如果有一致的体验，可以让新员工更快、更容易地上手，使他们能很快做出贡献。

一方面要确保一致的体验，另一方面还要允许员工定制他们的工作站和工作习惯，从而最好地满足他们自己的个人需要，在这二者之间要达到一种平衡。太多的定制可能导致知识的隔离，或者需要花费额外的时间和精力来处理特殊化的环境设置。但是近些年来，员工对于能够以他们自己的方式工作越来越重视，所以如果不允许员工发现和采用对他们最适用的方式，在聘用和保留人才方面就会是一个竞争劣势。



要有一个共享区提供本地开发环境的文档。这些文档可以存储在一个版本控制仓库、内部wiki或者甚至Google Doc中。随着时间的积累和使用经验的增加，人们会对这些工具越来越熟悉，所以这里的目标不是在文档中详细描述每一个细节，而是要让人们能够在这个环境中顺利地进行开发。

## 版本控制

如果能够提交、比较、合并和恢复仓库中以前版本的对象，会在团队内部以及团队之间更好地进行合作和协作。通过建立一种方法，使生产阶段的对象能够回退到之前的版本，可以使风险最小化。

每个组织都应当实现和使用版本控制，并在这方面进行培训和测评。如果多个人同时处理同一个文件或项目，利用版本控制，团队就能够处理这个过程中产生的冲突，并提供一种安全的方法进行修改，如果有必要还可以回滚到前一版本。应当在团队或产品的生命周期中更早地使用版本控制，这样能帮助养成好习惯。

为你的环境选择适当的版本控制系统（version control system, VCS）时，你要找的系统应当能够鼓励组织开展你想要的协作。鼓励协作的特点包括：

- 开放和创建仓库。
- 为仓库做出回馈。
- 规划对仓库的贡献。
- 定义贡献的流程。
- 分享提交权限。

有些工具尽管缺乏协作特点，但是在你的环境中已经使用了相当长的时间，所以对这些工具已经积累了丰富的经验知识。在这些情况下，要明确如果不迁移到另一个工具会有哪些影响（如招聘能力或合并不同分支花费的时间），然后考虑如果失去这些经



验知识又会带来哪些影响，试着对二者进行比较。如果有适当的流程，即使采用不太具有协作性的工具也能实现协作，只不过并不容易。



代码行数并不是一个准确的价值度量指标。有很多不同类型的开发人员，有些人会把数百行混乱的代码重构抽象为几十行可以轻松阅读的代码，使团队的其他人可以在此基础上更好地构建代码。另外一些人可能更关注查找代码中隐藏的bug。要适当地使用定量度量手段作为引导，鼓励你希望看到的行为。例如，除非能够从质量上检查代码，否则不要认为代码越多就越好。

与版本控制有关的其他术语包括：

**提交 (Commit)**

提交是一个动作集合，包括在版本控制下对文件做出的全部更改。

**冲突 (Conflict)**

如果两个更改本质上太过相似，版本控制系统无法确定哪一个是将接受的正确更改，这就会产生一个冲突。大多数情况下，版本控制系统会提供一种方法来查看和选择所需的更改，以解决冲突。

**拉取请求 (Pull request)**

拉取请求是一种允许个人发出信号的机制，表示可以审查一个贡献并合并到主干中。

**拣选合并 (Cherry picking)**

拣选合并是从一个分支选择特定的提交并应用到另一个分支的方法。如果你想从一个拉取请求选择特定的更改，这会很有用。

## 工件管理

工件 (artifact) 是软件开发过程中所有步骤的输出。在一个简单的仓库和一个比较复杂的完备仓库之间做出选择时，应当了解支持额外服务的成本以及固有的安全问题。

工件仓库应当具有以下特点：

- 安全。
- 可信。
- 稳定。

- 可访问。
- 版本化。

如果有一个工件仓库，你就能静态地处理依赖库。可以存储一个版本化的常用库作为独立于软件版本控制的工件，使得所有团队都能使用完全相同的共享库。二进制版本应当构建一次而且仅一次（尽管原本同一个二进制版本可以再次构建），这样有助于降低复杂性，可以确保测试周期中使用同一个二进制版本，并确保不同构建版本的提升。

利用工件仓库，可以根据使用方式存储工件。有些仓库系统一次只存储包的一个版本。这可能会在描述包的历史时带来问题，所以可能还希望增加包存储的重复因子，为工作流中的每个环境分别维护一个单独的工件仓库。

在一个组织演进的早期阶段，可能还不需要满足某些安全合规性需求。不过，随着沿生产线的发展和演进，这种情况可能会改变。有一个专用的本地工件仓库能支持更平滑地迁移，从而满足这些需求。

理想情况下，与你的环境中的其他构建和部署机制一样，本地开发环境也同样能访问内部工件仓库。这有助于尽可能消除由于生产环境与本地开发环境中使用的包和依赖库不同而导致的“在我的笔记本电脑上都能正常工作”综合症。如果限制或不允许访问，就会促使人们寻求绕过安全和其他策略的新方法。



### 更早地定义策略

更早地建立监管过程，在你的环境和约束背景下促进协作。例如，明确谁可以推送哪些工件，并确定如何审查工件、建立许可和保证安全。这可以缓解过期工件的相关问题。

如果你的环境中不能访问互联网，所有一切都需要你来维护，包括通用的软件仓库、特定于语言的包服务器、依赖库管理等，还必须复制大量共享服务。不过，这样做可能有很多好处，这样一来，如果上游工件发生重大变化而没有相关说明，你的组织也不会受其影响，另外，外部的严重问题也不会带来内部问题。如果依赖于互联网提供依赖库，这意味着最终会由其他人控制你的构建版本的可用性和一致性，这可能是很多组织想要避免的。

与工件管理相关的其他术语包括：

依赖管理 (Dependency management)

依赖管理涉及一个软件项目对另一个软件项目的依赖方式和程度。要提供这种依赖性，存在不同的机制。在工件层次，为软件存储依赖工件会很有帮助。

锁定 (Pinning)

锁定方法是为一个环境锁定某个工件的一个指定版本。在依赖管理中，如果要定义适用于项目的一个依赖软件工件的特定版本，这种方法会很有用。

推广 (Promotion)

推广方法是要选择一个软件的特定版本转向交付，其关键通常是成功地通过测试。

## 自动化

我们要提升产品的质量以及结果的精度和准确性，为了达到这个目标，自动化工具可以减少为此投入的人力、精力和/或材料。

## 服务器安装

服务器安装是配置和设置各个服务器的自动化过程。HP和Dell等硬件制造商都提供了适用于其品牌硬件的安装工具。

有些Linux版本也提供了特定于操作系统的工具。例如，Cobbler和Kickstart可以用来自动完成Red Hat Enterprise Linux或CentOS的服务器安装。运维人员可以编写Kickstart文件，指定硬盘分区、网络配置以及要安装哪些软件包等。



### 硬件生命周期管理

每个公司必须采用某种方式来完成硬件或生命周期管理，不过，云和基础设施或平台服务的出现在某种程度上减弱了对这个需求的关注。硬件生命周期从规划和购买（或租用）开始；然后是安装、维护和维修；以折损、返回或回收结束。

## 基础设施自动化

基本说来，基础设施自动化是指通过代码来提供基础设施元素。可以认为这个代码就类似于软件的其他部分，只是能够通过数据备份、代码仓库和计算资源恢复业务。



与基础设施管理相关的其他术语包括：

### 配置漂移 (Configuration drift)

配置漂移是指服务器一段时间后相对于其期望配置发生改变或漂移的现象。

### MTBF

MTBF是平均故障间隔时间 (mean time between failure)，即两个故障状态之间的正常运行时间。

### MTTR

MTTR是平均修复时间，恢复系统操作所花费的时长。

### 可用性 (Availability)

可用性是一个常用的度量指标，即一个系统或服务真正可用的时间与应当可用的总时间之比。可用性 =  $MTBF / (MTBF + MTTR)$ 。

### 容量管理 (Capacity management)

容量管理过程用来确保基础设施和其他资源有适当的规模，可以采用一种成本有效的方式满足当前以及将来的业务需要。

### 雪花服务器 (Snowflake server)

雪花服务器是一个通过多次人工更改达到其当前期望配置的服务器，这通常是一个组合，包括命令行操作、配置文件、手动应用的补丁，甚至还包括GUI配置和安装。



人们谈到运维角色中的配置管理时，通常是指基础设施自动化。取决于具体的系统，可能要对系统的不同部分完成自动化，包括服务器上要安装哪些软件包、使用这些包的哪些版本、系统上是否有各个文件，或者要运行哪些服务等方面。

“可以认为基础设施代码就类似于软件的其他部分”，这意味着可以利用一个公共的本地开发环境开发这个代码，通过版本控制确定版本，在一个工件仓库中确定工件的版本，测试并在真正投入生产阶段之前完成校验。

基础设施自动化至少应当提供：

### 配置漂移管理

配置漂移可能是因为手工更改、软件更新或错误或者数据不一致造成的。作为一

一个好的解决方案，应当有办法避免这种现象，通常会有一个单独的节点，定期根据期望配置检查实际配置，并自行修正所有不一致的方面。

### 消除雪花服务器

基础设施自动化解决方案可以避免创建雪花服务器，确保所有更改都是明确定义的。要消除雪花服务器，可以一次增加系统中一个部分的管理，直到可以使用同样的配置管理策略从头创建具有期望配置的服务器。

### 版本化基础设施代码工件

一个好的基础设施自动化解决方案会使用版本控制和一个工件仓库。这可以确保定义服务器配置的代码能够版本化，从而得到版本化的所有好处，如可以很容易地回滚到一个已知的正确版本，或者可以有一些后提交挂钩（post-commit hooks），能够对基础设施定义代码运行测试。这也是我们熟悉的一个过程，会让所有团队成员乐于为改进基础设施代码做出贡献。

### 尽量降低复杂性

通过根据平台类型或版本来指定配置版本，基础设施自动化解决方案允许个人（可以是任何角色）以最小的开销管理一个异构环境。



即使在一个系统数量很少的创业公司中，也不要建立额外的技术负债，这一点至关重要。可以投入一些具备运维技能的人员，他们了解雪花shell脚本与基础设施自动化之间的差别，这会带来很大的不同，使你能专注于你的主要竞争领域，而不必在此之外花费时间和金钱。

如果可用的基础设施自动化工具没有考虑到你的具体需要，可以扩展这个领域现有软件的基本特性或可靠性，与从头创建你自己的工具相比，这仍然是更加成本有效的做法。

利用基础设施自动化，我们可以得到可重复、一致、可记录、可审计的弹性过程。这会节省时间、提高人员的效率，支持更大的灵活性，而且有利于风险测评。基础设施自动化还可以增加自信度，我们能相信机器的设置和部署是一样的，这会减少根据系统差异调试问题所花费的时间。



可以对基础设施自动化与对一组服务器中每台服务器的手动配置做个对比。人类在完成重复性任务时总免不了会犯错。可能老系统上没有配置某个更改，或者遗漏了任务清单中的某个步骤，这就导致系统会有不一致的配置。

不要制定更多流程和任务清单。实际上，应当确保分配足够的时间将这些手工清单转换成计算机可执行的脚本。计算机比人类更擅长完成重复性的任务。

由于“使用基础设施作为代码”有很多切实的好处，公司在实现文化转换时，这通常是最先调查的工具之一。只有在具体使用工具的特定环境中才能理解这些工具，也就是说，环境的特定文化和理念会影响工具的功效。对你来说，哪个基础设施自动化工具最适用取决于你的特定需要。

## 系统供应

公司原先必须规划、购买和供应数据中心的硬件，不过如今它们可以选择采用云基础设施。利用按需计算，公司可以只购买它们真正需要的基础设施，并根据需要进行扩容或减容。与物理硬件相比，这个基础设施可以更快地购买和供应，而且对组织来说，这通常有更高的成本有效性。

系统供应是对基础设施自动化的扩展，使得公司可以根据依赖系统集群来定义他们的基础设施，而不只是单个节点。人们可以指定希望如何供应一组服务器，这只需指定一次，以后需要时可以任意多次地自动按照这个规范来供应。

## 测试和构建自动化

在最早的计算机和编译器时代，程序往往包含在一个源文件中，很少涉及多个源文件。随着程序规模和复杂性的增加，开发人员开始把它们分解到多个源文件中。某些编程语言的用户还可以使用一些标准代码库，这也增加了复杂性。由于有太多不同的源文件需要一起正确地编译才能得到最终的程序可执行文件，所以自动化构建过程变得很有必要。

如今的构建自动化工具通常会指定如何构建软件（需要完成哪些步骤，以及这些步骤的顺序如何）和需要哪些依赖文件（为了成功地完成构建，需要提供另外的哪些软件）。有些工具尤其适用于采用某些特定编程语言的项目，如Apache的Maven和Ant，尽管从理论上讲它们也可以用于其他语言，不过还是在Java项目中最为常用。另外一些工具，如Hudson或Jenkins，则可以更广泛地用于更多项目。

这些工具通常可以划分为以下3类：

### 按需自动化 (On-demand automation)

这个工具由用户运行，通常根据用户的决定在命令行上运行。例如，开发人员可



能在本地开发阶段手动地运行一个Make脚本，确保在正式迁入版本控制之前能够顺利在本地构建这个软件。

### 调度自动化 (Scheduled automation)

会根据一个预定义的计划（调度）运行这个过程，如每晚构建。每晚构建会在每天晚上创建，通常是没有人在工作的时候，这样在软件构建时不会发生新的变更（不过，由于团队的分布越来越全球化，情况可能已经不同了）。

### 触发自动化 (Triggered automation)

在指定事件发生时运行这个工具，如一个持续集成服务器，每次在代码中迁入一个提交时运行一个新构建。

## Yvonne Lam：测试、监控和诊断的定义

测试 (test)、监控 (monitor) 和诊断 (diagnostics) 这几个词通常会混在一起，在团队内部和团队之间带来很多混乱。为了更好地一起工作，团队需要建立一个共同的术语表来传达信息。这样可以鼓励共享知识，而不会限制任何团队成员，也不要要求所有人都了解每一个细节。

在Sysadvent 2014上，Yvonne Lam指出：一个团队在建立关于测试、监控和诊断的共同上下文时，应当问这样一组问题：

- 它要在哪里运行？
- 它要在什么时候运行？
- 它多长时间运行一次？
- 谁会使用这个结果？
- 该实体如何处理这个结果？

Lam进一步给出了一组定义，可以用来澄清它们的区别。测试 (Tests) 针对非生产系统运行，以考察系统或软件的准备情况。通常会在发生某个变更时运行测试。监控 (Monitors) 会按计划针对试生产和生产系统运行。监控通常会频繁运行，或者由事件触发。发生某个事件时根据需要针对生产系统运行的则称为诊断 (Diagnostics)。

要想最有效地跟踪不同系统的表现，这三者都是需要的，不过，如果能够明确不同团体或个人的职责领域稍有不同，这对于建立共同的认识和职责会很有帮助。

与测试相关的其他术语包括：

#### 冒烟测试 (Smoke testing)

冒烟测试最早源自于硬件测试，用来确定对一个设备加电是否会导致它开始冒烟，如果冒烟就说明存在严重的问题。在软件中，冒烟测试是一个非常基本的快速测试，用来确定输出是否可行和合理。

#### 回归测试 (Regression testing)

回归测试验证对软件的修改不会引入新的bug或故障。

#### 可用性测试 (Usability testing)

可用性测试需要针对用户测试一个产品，通过结合具体用户的测试来评测其功能是否满足预期要求。

#### A/B测试 (A/B testing)

A/B测试是一种试验方法，采用这种方法时，会对一个Web页面或应用的两个不同版本进行比较，确定哪一个表现更好。

#### 蓝绿部署 (Blue-green deployment)

蓝绿部署是一个发布过程，这里会维护两个相同的生产环境。可以认为其中一个环境是当前运行应用的环境，提供所有业务流。测试新版本的最后阶段在第二个环境中完成。测试通过时，业务流将路由到第二个环境。这种部署过程可以降低风险，如果新版本出现某种问题，可以立即回滚到前一个生产环境。

#### 金丝雀过程 (Canary process)

过去，挖煤工人用金丝雀作为早期警报系统来检测有毒气体。当金丝雀开始表现出中毒症状时，挖煤工人就能知道煤矿的状况不安全。在软件中，金丝雀过程是指在一个小的生产系统子集上运行新代码，查看这个新代码相对于原代码的性能如何。

## 监控

监控是一个涉及面很大的主题，包括很多方面，最常见的两个方面就是事件和分析。信息收集方法包括指标和日志。监控包括收集基本的系统级指标（如服务器是否打开，使用了多少内存和CPU，每个磁盘的容量使用情况等），还包括一些更高层应用的监控，如一个Web服务器处理多少个用户请求，队列系统中有多少项在排队等待，加载一个给定的Web页面需要多长时间，以及哪些数据库查询的运行时间最长。尽管

原先这只是系统和网络管理员涉足的领域，但随着软件越来越复杂，团队协作越来越多，人们开始认识到监控是产品健康的一个核心体现。

一般来讲，监控是跟踪系统和环境当前状态的过程，其目标通常是检查系统是否满足期望状态所需的一些预定义条件。监控、警报和测试往往混在一起，这会导致我们对所要完成或构建的目标产生混淆。前面已经提到，监控通常会按一个预定义的计划运行，而测试的运行往往是为了响应变更。警报是自动传送的消息，用来通知人们某个测试或监控的结果。

## 指标

指标是一组定性和定量的度量结果。一般地，会把指标与一些基准或既定标准进行比较，通过跟踪指标来完成分析，或者出于历史原因进行跟踪记录。职能组织中的指标通常自成孤岛，这对于正确地选择产品开发方向会有不好的影响。

指标是监控中一个很关键的部分，即使是最复杂的Web软件，也可以对几乎任意一部分收集和存储数据，不同的团队在工作中可以跟踪和使用不同的指标。StatsD和Graphite是跟踪、存储和查看指标的一个很常用的强大组合。



目前有一项社区推动的工作，即在GitHub上的一个指标目录仓库中定义系统和应用指标集，要按协议、服务和应用分组，这个指标目录仓库由Jason Dixon维护，他是Monitorama监控大会的组织者。对于想要建立或扩展指标集合的人来说，这个仓库可以提供一个很好的参考。

## 日志

日志是基于操作系统或软件消息来生成、过滤、记录和分析系统中出现的事件。跟踪一个软件问题的来源时，工程师首先要做的事情之一就是检查日志，查看是否有相关的错误消息。日志往往是一个宝藏，其中包含大量有用的信息，而且随着存储器变得越来越廉价，几乎可以保存和存储你想要的任何日志，以便以后使用。日志可能来自你开发的应用，或者你使用的第三方工具，甚至可能来自操作系统本身。由于不同软件之间没有日志标准，可能很难对日志中的事件进行分类来发现问题模式。

一个系统一天可以生成数百甚至数千行日志。在现代环境中，可能有数十个应用运行在数百或数千个服务器上，日志数据量非常庞大，不再是搜索一个日志文件那么简单。开发应用时要做很多工作来处理日志的存储和搜索。日志记录的解决方案非常复



杂，这已经超出了这一章的范围，尽管这一章不介绍这个内容，不过绝对不能低估日志的价值。

## 警报

监控和警报从性能方面来看很重要，不仅如此，从预防的角度而言也非常重要，因为它们可以帮助你找出潜在的问题，而不是等这些潜在问题最后发展成为客户面对的实际问题时再去考虑补救。例如，2013年10月某国HealthCare.gov网站启动时，原先就没有监控或警报机制来告知这个网站是否真正上线（本书写作时还是处于即将上线的状态）。

某国数字服务行政官Mikey Dickerson在多次行业讨论中指出，在最初几个月的自动化阶段，他的团队采用的监控形式简单到只是关注新闻，如CNN关于这个网站是否出现问题的报告。尽管警报不是万能钥匙，但一个成熟的警报策略确实可以减少困扰，否则问题真正摆到公众面前就很难堪了。

分析警报时，需要考虑以下几个因素：

### 影响 (Impact)

不是所有系统都有相同的影响。有些问题很广泛，会影响多个系统或很大的客户群体，而有些问题只影响到少部分系统和人，与这些问题相比，前者的影响会大得多。有些问题客户可能根本不会遇到，或者所影响的系统有足够的冗余，所以不会有太大影响。为了避免警惕疲劳，警报应当仅限于影响最大的问题，这个内容后面还会更详细地讨论。

### 紧急性 (Urgency)

与影响类似，并不是所有问题都同样紧急。紧急问题是指需要快速做出响应（或者有时需要立即响应）的问题。例如，你的网站完全瘫痪了，现在你正在损失钱或失去客户，相比之下，如果只是不能访问一个纯粹提供信息的博客网站，前者要更为紧急。对于哪些问题是紧急的，不同的利益相关人很可能有不同的观点，所以在配置监控和警报时要考虑到所有利益相关人，这很重要。

### 相关方 (Interested party)

总的来讲，对一个事件感兴趣的各方往往是会受其影响的人，这可能是你的客户（或者一部分客户），或者对于内部服务事件，相关方可能是一组员工。相关方还有可能是负责对一个事件作出响应的人。例如，如果只有数据库管理员（DBA）能处理一类特定的数据库问题，更有意义的做法是向DBA发出警报，

而不是向运维团队告警，因为运维团队唯一能做的可能也只是打电话给DBA。

### 资源 (Resources)

要响应一个给定的事件或警报，需要哪些资源？它们的可用性如何？是否有足够的人力来确保可以响应多个事件，或者是否只有一个人待命（这可能成为一个故障点）而没有任何备份？你的组织是否有必要的资源，当没有某个给定服务、硬件或某个人时也能正常运转？这些都是在建立警报时要考虑的问题。

### 成本 (Cost)

监控和警报存在相关的一些成本。这包括监控服务和解决方案的成本、历史监控或警报数据存储空间的相应成本，以及向人们发出警报的成本，另外从响应者的角度来看，当然还包括响应事件相关的成本，以及如果一个给定服务不可用而使公司产生的成本。

总而言之，警报是由通过监控收集的数据创建事件(event)的过程，其目标通常是让人们直接注意到需要人为干预的某个问题。

## 事件

事件管理是监控的一部分，系统和服务会受到事件的影响，事件管理就是要处理关于这些影响的现有的知识。对于全天候的24小时/7服务，这通常意味着需要基础设施中所有不同组件状态的有关实时信息。会配置一个系统来监控某个特定指标，或者根据一个确定的事件记录日志，如果达到某个阈值或满足某个警报条件则发出信号或警报。

由于现在很多软件开发都在Web软件上完成，所以希望这些Web软件能够全天候可用，另外因为工程师可能在家而不是到办公室工作，所以需要更多地考虑处理这种情况下出现的警报。要处理这个问题，一种方法就是尽可能地建立自动化事件处理。

很多警报和监控系统都提供有内置的方法，可以自动响应一个给定的事件。例如，Nagios监控系统就有“事件处理器”，可以针对不同的警报条件配置事件处理器。这些处理器可以做各种工作，从自动重启一个崩溃的服务，到创建一个ticket让技术人员替换一个出故障的硬盘，都可以配置事件处理器来处理。自动化事件处理器可以减少运维人员的工作量（可能在休息时间被叫起来处理问题的次数也能减少），不过自动化事件处理器也不是没有风险。重要的是，要确保故障条件已经明确定义，另外已经充分理解事件处理器过程来实现自动化，而且有必要的安全防护，以避免自动化带来更多问题而不是解决问题。

任何一个警报系统都不能保证100%的情况下100%正确。误报（false positive）是指本来没有问题，却生成一个事件。如果你的事件会生成警报，比如在人们休息时发短信把他们叫起来处理这些问题，误报就会不必要地干扰人们的正常睡眠。另一方面，漏报（false negative）则是出现了一个事件但是没有为它生成警报，这可能导致需要更长的时间才能检测 and 解决这个问题。误报和漏报都存在成本，哪一个更好或更糟取决于你的特定问题和环境。



一段时间后，随着你对问题和事件的实际影响有了更多的了解，你可能希望优化和调整你的监控和警报。我们建议监控警报的趋势，包括是否对每一个事件采取某个动作、总体来讲多少个警报可以采取动作及有多少个警报在休息时间生成等信息。

警报设计即如何创建警报，从而以最高效的方式传达信息以便人们理解，这是近来警报方面的一个很重要的主题。Etsy开发了他们的OpsWeekly工具来支持这种事件跟踪，另外可以按警报和组件的类型对警报分类。跟踪警报趋势和对警报数据完成分析会大大提升警报的有效性，同时对于需要对警报做出响应的人，还可以改善他们的健康和幸福程度。



就像其他第一目击者一样，现场体验会获得一些隐性的知识，如哪些警报是噪声。可能很难归纳出一个能清晰地处理所有情况的自动化机制，不过要不断努力改进警报系统的效率，这很重要。警惕疲劳或者对警报（通常是误报）不敏感都会导致对于真正的问题响应减慢，而且还会造成倦怠。

环境在改变：可能原先某个方面有问题，不过由于软件功能的某个改变，现在这个方面已经不再是问题，或者由于复杂性的增加，解决问题的老办法不再可行。人类可以改变，很快地处理这个问题，但是算法并没有同样的适应性行为。处理这种持续的改变是警报和事件管理的一个重要部分。

## 生态系统的演进

随着时间的发展，从服务器安装的自动化到基础设施配置和自动化，可以看到这里的趋势就是简化和消除可能导致人类犯错的重复性任务。随着容器的引入，从开发到生产的流程已经进一步简化。



在为环境的不同部分增加自动化的过程中，已经发现了一些新的模式。对于基础设施自动化，坚持采用一个操作系统的某个版本已经不那么重要，因为从安全的角度来看，在一个新系统上快速启用带更新包的新实例可能更有用。

持续交付和持续部署可以把人们解放出来，去关注真正重要的问题。通过带测试的构建自动化，可以缩短反馈周期，使我们更有信心，而且对系统也有更多的认识。

随着应用开发要求更多的可操作性，这个生态系统还会继续演进。手动开发12要素应用 (<http://12factor.net>) 与在服务器上手动配置道理是一样的。可操作的需求得到标准化和自动化时，员工就能够自由地选择语言和框架。

这些趋势使我们开始关注这样的工具：它们强调“我们”而不是“我”，会在团队之间建立共同的理解，鼓励把时间用在有意义的结果上。

## 小结

这一章中，我们概要介绍了工具当前的生态系统。尽管工具是devops环境中很重要的部分，但要强调它们会提升但并不能替代这个环境中的人文方面，这一点很关键。工具如何使用以及使用的难易程度会影响文化中特定方面的接受度和扩散。我们谈论devops工具时，是指工具本身和它们的使用方式，而不是其基本性质。

devops文化是团队、组织和行业间的一种协作。开发解决方案时，要考虑它们对团队和组织的影响，而不只是对个人的影响，这很重要。有时这意味着要调整期望，明确什么才真正对组织有好处，努力寻找适合更多人的解决方案，而不只是适用于组织里的那些“明星”或最有发言权的人，所寻求的解决方案应当对组织整体有积极的影响。

devops工具强调“我们”而不是“我”，允许团队和组织建立相互理解来完成工作。你选择的工具就是共同语言的一种选择。这个语言是对你的组织整体有好处还是只对特定团队的一部分有益？有时，由于没有一个均衡的工具，可能必须做出某种选择，使某个团队相对于其他团队有更高的认知成本。要注意所影响的团队的成本和同理心。

# 工具：文化加速器

工具是加速器，可以根据一个组织当前的文化和方向推动改变来加速。如果不了解我们当前的位置或方向，应用加速器会产生意外的结果，很可能有负面影响。

这个世界变化很快，一种直接的反应就是追随别人，希望复制他们的成功。我们需要花些时间来了解自己当前的位置，明确我们与其他团队、组织、竞争者和整个世界的关系。这会帮助我们确定当前应该做什么，哪些要延后，另外哪些要从我们的环境中清除。

上一章介绍了当前的工具生态系统，这一章会换一个方向，提供一些真实的例子来说明工具和文化如何相互影响和作用。这些研究并不是要提供特定的方法步骤，而是要展示组织可以在他们的环境中采用不同的方式评价、选择和利用工具。这些研究会指导并展示如何考虑工具的选择，而不是具体推荐一些特定的工具作为devops需要的完美解决方案。

## 工具对人的意义

很久以前人类就开始使用工具来更有效地完成工作。从打字机到字处理器，这些工具可以降低修改和纠正错误的成本。从打孔卡和汇编语言再到更高级的语言，这些工具则增加了对代码的理解。

发明这些工具的意义并不在于工具本身，创建这些工具是为了让使用工具的人更容易地完成特定的工作，这很重要，选择工具时一定要记住这一点。利用这些工具，我们可以更多地协作，因为软件不再只是由单个人编写和支持，现在往往要由多个人和多

个团队来编写，而且会由不同的团队来理解和维护，有时甚至要在多年之后由其他团队维护。

## 工具是什么？

讨论工具时，软件通常都会作为焦点，要使用哪些编程语言、IDE、文本编辑器、shell、配置管理解决方案或者聊天程序。不过，工具不只是软件。实际上工具是帮助我们实现某个目标的帮手，而且在这个过程中不会被耗费掉：

- 在一个数据中心，服务器升降车可以减少破坏，加速装架过程。
- 如果你要外出参加会议，或者要在数据中心里携带一个计算机，更小更轻的笔记本电脑可以少费一些体力。
- 选择一个硬件RAID解决方案而不是软件RAID解决方案时，尽管会花更多的钱，但是可以提供一些很好的特性，如电池备份和维护更容易。



### 最好的工具

工具的价值和成本是不一样的。如果都一样，我们就没有必要写这一章了，你完全可以选择满足需求的任何工具。即使是几乎所有人都认为很重要的工具（如配置管理或源码控制工具）也有不同，有些可能对某个特定的环境更为适用。

根据经验、知识和流程的不同，工具会有变化。这意味着，团队可以使用相同的工具，而得到的结果可能大不相同。工具必须适应环境的上下文。如果没有上下文环境，最好的工具就无从谈起，而上下文是不断改变的。

## 解决实际问题的适当工具

要让一个工具得到广泛的采用并实现成功的改变，它需要能够解决实际问题。人们要参与有关一个给定工具的整个决策过程，从而能发现、了解和解决他们的问题、困惑和担心。

一般来讲，首先要理解工具帮我们解决的实际底层问题，使我们能够选择适当的解决方案，并充分了解工作的复杂性。通过了解这些复杂性，我们可以努力将复杂性和相关的风险降至最小，相应地这会帮助减少开销，并确保人们专注于正确的领域。





在一些组织中，使用工具的人不参与采购过程。这可能会让一个流程或文化问题看起来像是一个工具或技术问题。

要让一个新产品或组织成长壮大，与维护一个现有的产品相比，需要考虑不同的问题。不能只由一个人根据他认为有意思和感兴趣的东西单方面地做出决策。通常之所以选择某个工具只是因为有人想尝试一下，这并不是一个充分的理由。

尽管使用工具可能很有趣，而且适当的自动化可以把人们解放出来，使他们有更多的时间和精力处理更复杂的问题，但是要让工具真正有效地发挥作用，需要基于更充分的理由选择工具，而不只是因为它们很有趣或者它们是新工具。

## 采用开源

开源社区为个人提供了一种与其他人协作的途径。员工要了解管理其他人贡献的意义，还要了解如何提供对其他人有用的贡献。例如，可以有多个分散的小提交，一次只关注某一个更改，这就比涉及很多不同代码的一个大提交更容易管理，也更容易接受。

公司是否要担心将工具开源会削弱他们的竞争优势？如果公司的成功依赖于工具，这应当是你的商业模式的一部分，你在靠软件本身挣钱，或者正如当前的很多公司一样，是通过对软件的支持或服务来挣钱。

为开源做出贡献最能体现公司的意图。在公司里将软件开源可以鼓励团队相互之间为对方的项目做贡献，而不是一切都从头做起，这也向个人贡献者和管理者展示了开源协作的好处。为开源做贡献和利用开源通常也是相辅相成的。习惯于开源社区的团队更有可能寻求已有的开源解决方案，而不是从头编写自己的解决方案。

很多公司都以devops领域知名的公司及其开源工作为标杆，如Netflix和Etsy，认为应当开始编写他们自己的工具并将其开源。尽管开源贡献有种种好处，但这里平衡也很重要：如果在另一个方向上走得太远，最后就会陷入“非我所创”综合症。这个词用来描述一些公司（原则上）拒绝使用第三方工具，因为这些工具来自公司之外，而非公司所创。

这种行为可能有很多原因，最常见的原因可能是因为竞争，公司不希望承认竞争者创建的解决方案，更不用说使用那些解决方案，或者公司对外部和未知的软件心存畏惧。他们可能不信任其他人，不相信别人写的代码能比得上他们自己写的代码，或者他们可能更愿意写代码，而不是去了解如何阅读和使用其他人的代码。有些人就是

喜欢这种挑战，去创建他们之前没有写过的软件，或者喜欢尝试用一种新语言开发项目。

关于使用第三方解决方案，有些担心可能是有道理的。如果要把它集成到一个现有的软件项目中，可能必须更新或改变这个项目的许可，使之与新的外部组件的许可一致。另外维护者将来有可能废弃这个软件，不再提供bug修正、安全更新或支持，或者将来的开发可能引入一些重大变更。

出于很多方面的原因，公司可能也不希望绑定到某个特定的开发商。不过，如果一味地强调“非我所创”而拒绝使用第三方软件，还是有很多缺点。除非你有一个安全专家团队，否则你编写的加密软件就有可能包含bug，从而存在安全漏洞。如果一个公司主业不是网络行业，编写他们自己的DNS服务器可能并没有好处。不论他们多么努力，结果也不太可能优于BIND，把时间花费在开发、维护和调试一个任何人都没有经验的软件上，这肯定是不值得的。



工具能够而且将会强化行为，这会影响你的文化。检查行为和文化时还应当检查工具，这一点至关重要，这里也体现了选择工具真正的重要性。采用开源不仅可以利用已有的工具和技术来提供新的解决方案，而且对协作、分享和开放的文化也有突出的影响。

由于有众多的开源项目和社区，鼓励个人做出贡献时，他们会获得大量经验，并了解适用于其环境的不同模式。

## 工具的标准

有效地工作最终可以归结为培养相互的理解，并妥善地协商，以克服团队力图同时达成多个目标时出现的不可避免的沟通问题。

工具可以用来：

- 改善沟通。
- 设置边界。
- 在devops组合范围内修复理解。

组织需要对工具进行标准化，对于完成相同功能的工具，标准化可以平衡支持这些工具的难度和成本。要通过标准化来增强组织，支持团队层次上的灵活性，以及允许个人选择自己的工具以保证敏捷和响应性，这些要如何做到呢？

## 工具分析的一致流程

在公司使用的技术发生改变时，工具的标准化可以在老工具和新工具之间搭建桥梁。通过对工具的评价、选择和退役采用一致的流程，组织可以做到：

- 确定一个满足大多数人需要的工具。
- 确保新工具中包含老工具中的必要特性。
- 确保员工得到适当的培训，能够有效地使用一个新硬件或软件。

如果没有一致的流程来建立必要的桥梁，员工就会对新工具或技术产生抗拒。一致的选择过程可以使风险最小化，确保当前需求的和新的需求都得到满足，有些人对环境中的改变比较抗拒，这种一致的选择过程也可以向他们提供保证。

一致性还有助于避免那些棘手问题，如果没有适当的流程或标准化，就可能出现这样一些麻烦。例如，如果每个团队都使用不同的问题跟踪工具或ticket支持系统，就会缺乏对整个公司的整体认识，存在更多的重复劳动，而且要花费大量时间让这些不同的系统互通互操作。

## 标准化的例外情况

标准化也有一些例外情况。如果一个团队有一些单独或特有的需求，就没有理由要求他们使用其他所有人所用的工具。

PCI合规性就是这样一个例子，这要求非常严格的责任分离。负责PCI工作的团队可能有单独的计算机，它们运行在不同于公司其他部门的一个单独的网络上。在类似这样的情况下，有一个隔离的环境允许使用不同的工具，而不会对组织整体产生有害的影响。需要根据具体情况做出决策。

尽管有很多共同点，不过每个团队和每个公司都会有一些特有的需求和经验。在这一章的案例研究中，我们将分析两个公司，了解他们如何选择工具和做出实现决策。虽然有一些惯用做法，但是根据环境中不同的技术堆栈，会有不同的工具选择。



## 工具不重要

工具是否重要？另外如果工具会带来影响，那么它们会有多大的影响？对于这些问题有不同的观点。一些开发商把一切都称为“devops”，并以此推销他们的产品，而不论这个标签是否准确，作为对此的回应，就产生了“工具不重要”的观点。

“工具不重要”有两层含义：

- 工具不是一个devops文化存在的充分条件。
- 工具不能修补被破坏的文化，工具只是展示和加强环境中的现有条件。

最终，如果一个“devops解决方案”只涉及工具，而不考虑组织中谁使用工具、如何使用以及为什么使用，这样的解决方案并没有真正掌握devops是什么以及devops为什么如此成功。不要试图只用工具和技术来解决人文问题。

## 流程的失败，而非工具

如果一个公司只是使用每一个都很独特的雪花服务器，而不知道如何实现和使用配置管理，这个公司很可能失败。如果无法对环境问题快速做出响应，这会导致宕机以至于损失收入。如果适当地使用配置管理，只要所选择的工具能完成你要求的工作，具体选择使用Puppet, Chef, Ansible, Salt, CFEngine还是某个尚未完成的新CM系统并不重要。



尽管工具之间存在技术差异，但最重要的是这些工具是否具有特定组织解决其问题所需的特性，同时有利于促进所期望的文化。

## 工具选择的康威定律

康威定律（Conway's law）的名字来自计算机科学家和程序员Melvin Conway，其思想是软件的开发方式往往反映了开发该软件的团队的结构和组织。这说明，要让两个软件组件协同工作，如果它们由不同的团队设计和实现，那么这两个团队也必须能够沟通。

反之，如果团队不能很好地沟通，如在一个严重割据的环境中，往往会创建根本无法协作的产品。对此有一个推论，团队选择和使用工具的方式与团队本身的结构和沟通

模式往往是一致的。如果两个团队本身不能相互沟通，他们不可能因为公司开始使用Slack作为新的聊天程序就开始沟通。

## 工具对文化的影响

由于工具会显著影响行为，在评估你的环境、评价文化和技术格局以及协同定义团队或组织的目标和愿景时，要仔细考虑不同工具鼓励或不鼓励哪些行为。要记住，这是一个持续的过程，需要不断重新评估。

### 工具影响沟通

工具会塑造行为，所以如果有一些工具能够减少与其他团队沟通中出现的摩擦，就更有可能进行沟通。如果一个公司甚至没有任何聊天软件，或者他们使用的聊天软件存在一些技术局限性，不允许团队间交流，沟通就会困难得多。



在谈到工具时，我们要强调沟通的重要性，因为沟通可以促成或破坏一个环境中的合作、协作和亲密性。不论你在讨论专门为沟通设计的工具（如聊天解决方案），还是那些工作流和使用中涉及沟通的工具，都是如此。

很多情况下，如何使用工具比选择特定的工具更重要。以ticket支持系统或bug跟踪系统为例。如果每个团队都认为工具不重要，并选择一个特定的ticket支持系统作为其工作方式的补充，在这个过程中会有大量变化，由于有不同的特性和实践做法，不同团队的成员要想一起高效地工作会极其困难。人们可能要管理更多帐户，或者无法了解另一个团队的工作(缺乏可见性)。



缺乏可见性是经常困扰孤岛组织的问题之一。孤岛化可能导致重复的工作，不清楚所做工作的细节（或者根本不了解是否在开展工作），另外会导致团队之间的不信任。

由于工作社区建立在有效的沟通之上，一定要记住工具以什么方式增进或妨碍沟通，这很重要。为了能够协同工作，沟通是很关键的一部分，公司使用什么工具和流程来实现人际沟通会对文化产生显著的影响。另外，每个工具的用法都有隐含有相应的沟通途径。

在第二部分中提到过，关于沟通需要考虑很多因素。这些因素并不包括寻找一个沟通工具来满足一个健康组织的所有内在需求。另外，沟通需求往往会随着公司的发展而改变。在一个小创业公司中，每个人都能很容易地与人交谈，所以可以通过聊天来沟通，这是合适的，但是经过一段时间后，你可能会发现团队逐渐会转向使用email或团队wiki。



### 度量参与度

随着公司的成长，要了解和度量公司更大范围内的参与度，这一点至关重要。

要在适当的时间找到适当的工具，这是一个迭代的过程。要确保在重要的谈话中能听到所有意见，这有助于建立一个更健康的公司。不能认为沉默就表示大多数人都同意。对一些优秀团队的研究表明，如果团队中每个人都能发出声音，这样的团队更高效也更有生产力。<sup>注1</sup>

与远程办公的员工一起工作时，要投资建设高质量的视频会视解决方案。确保你的团队成员有高质量的耳机，因为笔记本电脑自带的麦克风和音箱可能效果不好，会导致人们不愿意使用这个媒体来交流。这会使远程工作的员工感觉被孤立，无法参与重要的谈话或决策，因此往往不会很努力地工作。

关键是要基于沟通本身的内容、紧急程度、上下文和其他因素来选择工具、平台或方法。一旦根据你和你的团队参与的沟通类型明确了这些需要，可以根据其他需要选择特定于这种沟通类型的工具（如付费购买一个聊天程序或使用一个免费程序）。

### Hollie Kay：作为一个失聪的开发人员<sup>注2</sup>

当我还是婴儿的时候就失聪了。不是太严重；我的听力损失为中度到重度，主要是在较高频段有问题，而大多数人说话声音都在这个频段。我靠唇读和识别元音模式来理解人们的语言。识别以下内容会特别费劲：

- 辅音，特别是齿擦音和清辅音（所有辅音都是高频音，而且清辅音和齿擦音不会引起声带振动）。
- 句子开始（和结束）。

注1： Anita Williams Woolley, et al., “Evidence for a Collective Intelligence Factor in the Performance of Human Groups,” Science, October 29, 2010.

注2： <http://cruft.io/posts/deep-accessibility/>.



程序员往往给人留下一种刻板印象，通常被看作是害怕与人相处的孤独怪人，这种成见不公平，也不正确。作为一个团体，我们是一个非常社会化的集体。我们写博客，我们在大会上发言，我们开发教程，我们还会热心地指导别人。这并不是新现象，这种气氛可以追溯到互联网时代的早期，那时贝尔实验室或MIT以及众多其他研发组织中就是如此。我热爱这种社会化的代码世界，因为你的周围有一群能干而且充满热情的人，这很重要，会促使你成为一个更好的开发人员。不过，有一点我总是退避三舍，这就是结对编程。

理论上讲，结对编程很棒，就像增强版的橡皮鸭调试。你与另一个比你懂的更多的人一同工作，他能指导你，或者这个人比你知道的少，很愿意接受你的指导，也可能他和你水平相当，可以与你一起摸索解决方案。另外，要知道这很有趣。你可以了解你的同事。你会意识到每个人都会犯错误。在你部署不该部署的代码之前，需要有人阻止你。

不过，如果你失聪，这种动态变化和乐趣就会变成折磨。对我来说，结对谈话比没有还要糟糕。作为结对编程中的驾驶员（driver），我要想着代码、类型，还要同时看着我眼前的屏幕，读身边结对伙伴的唇语，理解他们的高频口语和技术行话（通常没有上下文），可能只有约30%的正确率，这真是痛苦。最终，不仅我开始情绪低落，身边的领航员（navigator）也越来越崩溃，直到最后我完全放弃，干脆换他来作驾驶员，因为这是我们能继续推进的唯一方法。如果我作为领航员，情况会更糟糕，驾驶员要经常看着他们的屏幕，因为很难在考虑如何编写代码的同时还要考虑结对伙伴的沟通需要。我知道！我当然知道。所以我成为一个消极的领航员，驾驶员要完成所有工作，这对任何人都没有好处，真的很糟糕。太糟糕了！

所以有机会在Pally项目中与Rowan Manning结对真的很棒，这是为Nature构建的自动化可访问性测试工具。使用Screenhero建立一个远程结对会话意味着我们都可以看着屏幕并使用文本交流，不会丢失任何信息，也不会产生任何误解。这是我第一次正常地顺利完成一个结对会话。这种感觉难以言表，因为我想对于大多数听力正常的人来说，很难体会与一个失聪的人交流时会丢失多少信息。想象一下，在你的城市里你读到的所有书中60%的文字都被随机地抹成空白。然后再想象当你度假来到旁边的另一个城市，（很幸运）那里没有人做这种事情，突然之间，你可以读到完整的书，不再需要做任何猜测。就是这种感觉。

## 工具影响更宽泛的行为

这个原则不仅适用于ticket支持系统，还适用于基础设施自动化、聊天系统、部署工具以及组织中多个团队使用的任何工具。尽管明确每个人的需要并尽可能多地满足这些需求很重要，但是不太可能做到100%的人对所有工具都100%满意，肯定要有取舍。某种情况下，继续争论要使用哪个工具没有任何意义，可能最后还会导致敌对，因为这会浪费所有时间，而争执还会继续。更好的做法是，干脆选择一个工具，然后一致地使用这个工具。

记住这些警告，如果有众多选择都能满足你的需求，具体使用哪一个特定的工具并不重要。这一章不打算告诉你哪个工具才是真正的devops工具，因为并没有这种概念。这与宣称ed<sup>注3</sup>是编辑器大战中真正的胜利者是一样的。关于“最好的”工具，永远不会达成一般共识，最好的解决方案完全取决于要解决的特定问题。

## 工具的选择

要选择一个新工具增加到你的环境中，这可能是一个重大的决策，做出这个决定可能涉及很多人，他们往往都有自己的观点。下面是选择过程中要考虑的几个重要因素：

- 产品开发。
- 社区健康。
- 内部定制。

这里所列的并不全面，因为特性的可用性、预算以及与当前工具集或环境的互操作性也是要考虑的重要因素。

我们强调上面的这3个因素，因为它们是不同的组织共同的需要，另外这几个因素在其他地方都没有详细的介绍。不同的组织会有不同的需求，这会要求一些必要的特性、不同的预算，以及环境中各种已有的工具集。

所有这些因素对于选择过程的有效性都有重要的影响，所以一定要了解选择工具时有哪些决定因素。

注3：ed是UNIX的一个行编辑器。它曾经是系统的默认编辑器，尽管功能很强大，但是由于它的简洁性，使它很难在自动化中使用。

## 产品开发

一个积极开发的产品会更快地提供新特性，支持更新的操作系统和平台版本，并及时处理安全漏洞。如果使用一个未得到积极开发的工具，会导致更多的时间花费在处理bug或等待新特性上。

新特性多久会实现和发布？是否跟踪并定期评估这些特性请求来建立产品路线图？如果发现严重的bug或安全漏洞，多快能得到修正？

查看你考虑的工具的新版本。注意主版本和次版本日期、版本说明是否有用（如果你决定升级，版本说明中如果能给出特定的bug或ticket号，这会比简单的一行“bug修正”有用得多），以及更新过程是怎样的。

另外考虑你与产品开发人员能有多少联系。在开发这个工具的组织里，你有直接的开发人员或支持联系人吗？如果直接分配一些人与你一起工作，这意味着你能得到更好的支持，因为你们可以不间断地对话，这可以帮助确保你的问题得到解决，而不是消失在某个未知的ticket支持系统或email收件箱里。

## 社区健康

社区健康是对一个团体整体健康的度量，在这个团体中，人们通过共同的规范、价值观和行为联系在一起。社区可能围绕着某个特定的工具、一组工具和实践或者某个角色发展起来。

要注意作为健康信号之一的社区活动，包括：

- 对拉取请求的响应率。
- 解决问题的平均时间。
- 发布频率。
- 内容创建（博客贴子、文章、新闻）。
- 论坛交流率。

除了积极的社区活动，社区和相关活动还应当培养一种安全、尊重、协作和包容的环境。注意社区成员相互之间如何相处。考虑以下问题：

- 项目和社区活动有行为守则吗？



- 问题和拉取请求的讨论如何结束？

在一个社区中，如果人身攻击随处可见，或者容忍性别偏见等行为，这就不是一个健康的社区。这适用于所有社区，而不只是开源项目。在日常工作中使用这些工具的人们往往会与其他用户交互，可能会参加关注工具及如何使用工具的当地会议或更大规模的会议。



在这一章前面讨论过，开源软件的好处之一是你不必一切都从头做起，去解决其他人已经解决过的问题。如果一个开源解决方案有一个强大的社区提供支持，它的实现会更有效。

## 内部定制

如果一个工具很容易定制，这有助于提供一个健壮的解决方案，对一个环境的技术和人员方面都很合适。如果一个组织中很多人都使用这个工具，这一点尤其重要。倘若工具能很好地应对这种规模，就能随着组织的发展而发展，而且能更容易完成工程项目。

开源工具通常是最可定制的，因为你有代码，可以更容易地读代码，并根据需要进行修改。这对于完成bug修正等工作很有帮助，你不必提交一个支持ticket然后苦苦等待，而是可以做更多工作来找出bug，甚至可以自己提交一个包含修正的拉取请求。即使是非开源的工具，注意它们是否有类似API的特性可以用来开发额外的工具来处理或使用这个工具。

如果能够定制一个工具，修正你自己的bug，甚至增加新的特性和扩展，这会让工具越来越好用。如果你提供了一个很细小的特性，尽管不能广泛使用，但是对于你自己的某个团队可能非常有帮助，如果能自行增加这个功能，这就远远好于等着某个产品开发人员为你实现这个功能。有些工具只是让人能接受，而有些工具工程师会发自内心地喜欢使用，这两者的区别就在于这一点。

## 示例：版本控制的比较

版本控制系统会记录一组文件随时间的变更情况。CollabNet在2000年创建了Subversion项目，这是一个开源软件版本和修订控制系统，这个系统设计为与广泛使用的Concurrent Versions System (CVS)兼容。Subversion 1.0 (svn)于2004年2月发布。当时的技术和习惯决定了svn的用法和特性。svn架构的核心是一个集中仓库的概念。这个集中仓库允许用户控制谁能提交变更。

一年后，2005年发布了Git。这也是一个开源软件版本控制系统，其关注点是非集中式版本控制、速度、数据完整性以及对分布式非线性工作流的支持。这为每个开发人员提供了完全的本地控制。尽管可以采用一种集中式工作流，并建立一个“集中的”仓库，但是这些流程可以很灵活，你可以使用想要的技术，而不是必须由系统为你定义。尽管上线时间可能有点长，但是这个功能可以支持更快的组织调整。

## 示例：手动基础设施配置与自动化

大多数成熟的基础设施自动化解决方案总的功能都是类似的，不过它们的实现有所不同。对于所有不同类别的工具，每个工具可能会影响协作的不同方面。

在很多组织中，系统配置是一个手动过程。人们用一个清单记录这个过程并升级。如果漏了某个步骤，可能导致系统处于一个未知的状态，要做很大努力才能恢复。

Adam Jacob开发Chef软件时，他努力创建了一个能跨组织使用的解决方案。Chef可以提供配置和管理抽象，它创建了一种语言，允许个人用代码定义基础设施和策略。

要创建一种语言来支持开发人员、系统管理员、安全运维和质量保证工程师不同的观点，这会很困难。利用Chef，并不是重用原来的术语（不同的角色被赋予不同的优先级），Jacob创建了新的术语，包括resource和recipe。

要记住最重要的一点是：沟通工具就像我们在第2章介绍的攀岩例子中的同伴。他们可能是很好的伙伴，能帮助我们保持所建立的组合，或者也可能是不好的同伴，会让我们分心或者阻碍我们前行。当然，这取决于如何使用各个工具。如果试图使用一个紧急程度很低的媒介（如email）来完成需要立即响应的工作，这就会带来问题，就像相比于用言语来描述某个东西，可能在一个白板上画图会更快而且更有效。

最后，选择工具时，必须在内聚性与灵活性之间达到平衡。如果使用了太多的沟通方法，人们就必须完成大量搜索来找到他们需要的信息（信息在一个email里，还是在Google Doc或Confluence wiki页面上），或者要努力确定如何最有效向人们传达信息（要完成类似这样的工作，我应该发一个即时消息还是文本消息，或者是不是可以当面告诉他们）。另一方面，如果方法太少，又会带来麻烦，我们都听说过这样一些公司的故事，他们什么事情都要开会，而大多数情况下一个简单的email就能解决问题。

我们的组合重点在于有共同的理解，如果有一些传统或习惯，人们可以利用这些传统确定最有效的媒介，这对我们的组合会很有帮助，不过前提是要有足够的灵活性，可以为适当的工作选择适当的工具。

# 审查你的工具生态系统

除了选择要增加到环境中的新工具，还必须检查这个生态系统的状态。对于很多工具而言，第一项“石蕊试验”是要检查这个功能在你的环境中是否已经存在。审查环境来明确工具生态系统时，要包括谁能访问这个工具以及工具总体使用的有关信息。另外还要包括同一类中多个工具或环境中重叠工具的有关信息。这会指出哪些领域可以改进，为此可能需要额外的培训或替换工具。

流程要与个人使用特定工具的需求相一致，这对于有效的使用工具至关重要。太多的流程会增加个人的成本，因为他们要维护这些流程错综复杂的上下文。这些工作可能会让人们分心，而不能专心完成具体的项目工作。而流程太少又会导致另一个问题，由于工具以及使用工具的方法太多，这会使团队缺乏内聚力。这对个人也有影响。人们可能要花时间来修复理解，合并工作或者检查重复的工作。要找出一个合理的基准，这是明确和选择工具的关键。在组织规模扩大和缩小时这一点更为重要。

总的来讲，我们之所以使用工具，就是为了帮助创建适当的环境，使人们能专注于解决真正的问题。创建这种环境是管理一个成功团队很重要的一部分，而且这要求不断的努力。这种审查不是只做一次然后就抛之脑后。实际上，不论是个人贡献者还是管理者，都应当采用一种主动的方法确保构建和保持一个高效的环境。

## 工具消除

要定期审查当前流程和工具，确保它们仍然有效。自动化和技术负债跟踪可以帮助找出应消除的流程和工具。这有助于避免一些不应有的情况，如使用这个工具的人要为此做额外的工作，另外可以帮助找出和消除有相同作用的工具，从而降低成本，也可以减少混淆。

这些审查还应当包括与员工定期会面，来发现对他们及其工作有影响的情况。可以询问以下问题：

- 哪些事情让你兴奋/受挫？
- 哪些事情会补充/减少你的能量和动力？
- 对于你当前所做的工作，你认为有什么价值？
- 你认为你会带来什么影响？
- 哪些工作应当停止？



- 哪些工作应当开始?

要向平时大部分时间都在使用这些工具的人询问这些问题。关于如何选择和消除工具,应当由底层做出决策,也就是说,作为利益相关人,具体使用工具的人会比不使用工具的人更有发言权。这些决策不能由对工具没有任何亲身体验的人从上到下强制做出。

## 改进:规划和度量更改

持续的改变需要一定的时间。不论对于软件还是人,这些都是很复杂的问题,没有简便快速、十全十美的解决方案。“SMART”目标<sup>注4</sup>的根本原则指出:目标应当是特定的、可度量的、可达到的、实际的而且有时间限制。SMART更改对于工作组织和文化同样很重要。

要明确你解决的特定问题。做出一个更改之前,先要查看需要做什么。确定谁对这个项目感兴趣、谁有时间以及这个项目的总体价值。了解不同的观点,并明确可能的项目。确定项目的优先级,确保你确实在解决正确的问题。

将特定的项目分解为可以完成和解决的小目标。这些小目标通常更容易规划,相应地更容易考虑和确保它们是否可达到、是否实际以及是否解决了正确的问题。为了很好地规划这些项目,需要明确你在为谁解决问题。他们的需求和动机是什么?他们是否经常使用这个解决方案?要描述解决方案:重点强调最终目标。与利益相关人讨论,并确保他们采纳你的建议。这通常会花很多时间和精力。

一旦你知道了要解决的问题以及要为谁解决这个问题,接下来可以开始找出可能的工具。在确定采用某个工具之前,先要调查提出的各种不同解决方案的优缺点,要让利益相关人参与进来,使他们能帮助评估可能的解决方案。有时你可能发现现有的解决方案都不适用,必须自行发明和开发工具。尽管在预算方面内部开发看起来成本较低,但要考虑到支持长期开发所需的时间和资源。

最后,这些流程要不断迭代,在这个过程中要度量更改产生的影响,以确定你的解决方案是否可行。具体度量哪些方面可能会根据你要解决的问题稍有变化,不过关键如果没有度量,就无法判断更改的影响和有效性。

---

注4: G. T. Doran, “There’s a S.M.A.R.T. Way to Write Management’s Goals and Objectives” (Management Review, 1981)。

## 案例研究

这一章的第一个研究案例是DramaFever，这是一个流媒体视频平台，还运营着SundanceNow Doc Club记录片网站和Shudder恐怖片网站。DramaFever2009年成立，2015年年中完成这个案例研究时有大约120个员工，这个平台提供国际化内容，包括15个国家70个内容提供商提供的15000集电视剧，观看者达2000万<sup>注5</sup>。为了深入了解他们如何评估和利用工具和技术，我们访问了Tim Gross和Bridget Kromhout，做这些访谈时他们任DramaFever的运维工程师。

我们的第二个案例研究是Etsy，这里会延续第2章中Etsy的故事。Etsy由Rob Kalin、Jared Tarbell、Chris Maguire和Haim Schoppik在2005年创立，这是一个销售手工艺品和古董的全球在线集市。在2015年第二季度，Etsy有大约785位员工。这本书的合作者之一Ryn Daniels分享了她在Etsy的个人经历，此外，我们还与Etsy的运维工程师Jon Cowie进行了交流，来了解Etsy如何评估和利用工具和技术。

为了完成这两个案例研究，我们还通过已发表的博客文章、演示文稿和公司宣传材料收集信息。这两个案例研究都提供了一些例子，展示了这两家公司如何发现隐含价值，如何采用必要的实践方法，以及如何评估和选择工具来加入一个环境，不过要再次提醒读者，不要盲目地采用这些工具或技术，或者在没有深入地理解和考虑为什么使用这些工具以及如何使用的情況下不要盲目使用。

## 分析DramaFever

Tim Gross最初从事的职业是建筑而不是软件，后来转行到IT和工具开发，最后成为DramaFever的第一位“DevOps工程师”。尽管他的角色主要是运维，但是由于这个工程团队规模太小（只有一个人），这意味着他经常还要做开发工作。2013年3月，又聘用了第2个运维人员。

在这个团队中，并没有正式或明确的角色和职责划分，这只是一个逐步创建运维团队（ops team）的过程。尽管公司给他们的头衔是DevOps工程师，但这个角色基本上都是在做运维工作，在他们的招聘启示中就可以体现出这一点：“管理和自动化[……]基础设施的所有方面，包括网站部署、CDN和云服务管理”，另外还有一些特别的职责，需要维护高可用性生产AWS系统并随时待命。这里并没有特定的devops项目。

---

注5： Peter Shannon, “Scaling Next-Generation Internet TV on AWS with Docker, Packer, and Chef,” October 20, 2015, <http://bit.ly/shannon-scaling>。

DramaFever招聘启示包括以下声明：

我们着力帮助我们的工程师解决他们热衷的问题，如果工程师们认为可以，我们还会鼓励他们対现有架构的任意部分做出改进。

这个声明体现了一种实用的方法，会采纳和鼓励不同的观点来发现和解决问题。

2014年7月，Bridget Kromhout加入了DramaFever的DevOps团队。描述DramaFever技术堆栈时，Kromhout指出它完全在Amazon Web Services (AWS)中，有一个Django/Python Web应用，另外还有不断增加的Go微服务。由Akamai为他们提供内容交付和边缘缓存，Akamai是一个业界知名的内容交付网络（content delivery network，CDN）。

请求路径代码（这个应用代码声明了一个最终用户请求在代码基中经过的路径）以及所有关联服务有更高的可用性和延迟需求，远远超出其他应用的需求。请求路径使用不可变基础设施（这是使用Chef和Packer构建的），从2013年底以来，应用代码本身一直在Docker容器中运行。

Kromhout指出：

我们的应用代码存在于无状态实例中，一周时间中实例数会自动扩充10~20倍。我们的持久存储层由Elasticache (Memcached, Redis)、RDS (MySQL)、DynamoDB和Redshift提供。日志会送入ELK并用CollectD和StatsD写至Graphite。

不在请求路径中的服务包括异步Celery工作进程、cron作业、日志聚集和指标服务器（如Graphite或Logstash），或者诸如QA跟踪工具等内部应用。Kromhout继续指出：

尽管所有这些服务对我们的业务都很重要，但对用户并没有太直接的影响。如果一个cron作业没有触发，运维工程师可能花一个小时才能搞清楚为什么，我们可以留出这个时间，但我们的用户不会注意到这一点。不过，如果Django应用由于某种原因在服务区无法运行，我们的用户就不能观看他们喜欢的电视剧了！

## 现有技术的影响

从2006年AWS上线开始，这个行业发生了巨大变化。公司不再需要聘用具备数据中心管理技能的人员。他们需要的人员要具有管理共享工具服务方面的技能。DramaFever率先采用了AWS，并继续使用这个服务提供计算资源。Gross解释道：



我们使用Google App Engine(GAE)来完成一些小的一次性项目，不过只要这些项目开始得到更大量的使用，我们就希望把它们引入我们的AWS环境，在这里我们可以有更多的控制。

我们的图像处理微服务ImageBoss就是这样一个例子。这个服务可以根据需要调整和剪裁图像大小，利用这个服务，我们的创作团队不必再为每一个图片创建大量不同的版本。原先这个服务部署在GAE上，不过（当时）每个节点上只有一个内核用于Go，操作成本非常高。通过把它转入我们的AWS环境，我们就可以在适当时调整应用的托管环境。

尽管一般认为AWS比其他云提供商更昂贵，但DramaFever依赖的特性集是其他服务商无法提供的。当问到使用AWS的成本分析时，Gross解释说：

如果我们要从AWS转换到另一个提供商，就需要为SQS或DynamoDB之类的托管服务寻找替代服务，这差不多会抵消掉转换可能带来的所有好处。另外，我们的工作负载非常适合采用AWS的按时计费模式，我们会在一天内按某种规律对节点上下扩容，所以我们能合理地预测我们的需求，并能以较低的成本扩容来处理访问激增的情况。

基于这一点，与CDN成本和工程师工资相比，我们的AWS总账单完全是小巫见大巫（CDN成本要高出一个数量级，因为我们一个月要传送PB级的视频流）。如果转换到另一个提供商，尽管托管成本会有少许改善，但是会耗费我们大量的工程时间，所以可能并不值得。



在选择工具的过程中，考虑现有技术时，要问问自己：

- 哪些特性是绝对要有的，而哪些特性是可有可无的？
- 现在有哪些可用的解决方案，还有哪些很快就会有？
- 如何平衡你需要的特性和可用解决方案的成本？

## 新兴技术的持续影响

随着技术的演进和生态系统中不断出现新的工具，要了解是否采用某个特定的工具可能很有难度。对于一个小公司，由于有各种计划外的工作，技术债务不断增长，所以重要的是首先解决最有价值的项目。

Gross描述了2013年10月他的两人团队所面对的一些问题：

- 主Django应用部署很慢，而且是非自动的，总有一些复杂的故障。这个Django应用是一个关键的请求路径应用，有很高的可用性和延迟需求。使用git clone会使部署减慢，有时还会在部署过程中失败。
- 部署复杂性不断增加。不能采用现有流程部署新的Go应用。二进制代码和解释源代码有不同的交付需求。
- QA测试和生产部署流程是分开的，没有审计功能。
- 不同的开发和持续集成环境。

Gross还描述了他的团队希望达到的目标：

- 从一个整体应用转为较小的微服务。
- 开发和QA环境中在相同的主机上隔离应用的不同版本。

Gross解释说：

运维团队（那时只有我们两个人）根据一些指标（如问题与解决方案的适应度、我们对实现语言的经验、已知的故障模式等）评估了多种选择方案，最后选定了Docker。它承诺为我们提供一个通用的部署接口，可以解决所有问题。

这里存在一个很大的风险，那时Docker还不建议在线上环境使用（那是2013年10月，所以当时还是0.6版本），我们还不习惯把非生产项目部署到生产环境。我们知道，如果情况变糟，我们总是可以退回到更成熟的底层LXC。我们向开发团队的高级成员展示了这一点，来得到他们的认可。

在开发/QA环境中多次尝试后，我们将Docker部署到生产环境来运行我们的主Django应用。只有当所有问题都得到解决后，我们才开始将其余的应用部署到这个容器中。

任何技术的引入都会带来新的问题，必须解决这些问题。持续集成（CI）环境是指这个环境会通过自动化频繁地集成工作。每个集成会通过一个自动化过程进行验证，这个自动化过程会构建和测试所提交的代码，其目标是快速检测错误。一般的做法是启动一个环境，测试代码，然后撤销环境。

在他们的CI环境中，他们经历了一些与磁盘有关的问题，这是由于频繁的容器搅动造成的，为了解决这些问题，他们必须改变所用的Docker存储驱动（移植绝非易事）。他们还遇到Docker注册表的可扩展性问题。为了解决注册表的技术问题，最后他们在每个主机上部署了一个本地注册表，并辅以AWS S3作存储，而不是使用一个中心服务器。

将Docker引入本地开发环境时出现了第三类问题。Gross指出，

如果完全在AWS中部署，运维不会有太多麻烦。不过如果要在本地运行Docker，我无法确保能提供一个好的解决方案。由于开发团队总是忙于开发新特性的工作，他们无法投入时间来学习新技术。这意味着，我们提出采用`boot2docker`来完成本地开发时，最后会发现人们缺乏相应的培训，这个缺口很大，会持续很长时间，这不是我们希望的，而且这可能会带来内部磨擦。这是我们学到的一个经验，如果基础设施需要调整，完成设计时应当让开发团队更直接地参与其中。



不论是已有技术还是新技术，都需要一个仔细的选择和评估过程，这很重要。不过，对于新技术（不论是一般意义上的新技术，还是只对你的组织而言），要问问自己：

- 新技术有哪些已知的风险？
- 可能有哪些未知的风险？
- 你要解决哪些问题，而用现有技术无法解决？

## 亲密性促进实践方法的采用

无问责事后分析（post-mortems）的目标是创建一种持续改进的文化。

——@0x74696d at @dramafever

我很高兴，@0x74696d讨论@dramafever应该如何处理失败时引用了@codeascraft。感谢Etsy的帮助！

——Bridget Kromhout (@bridgetkromhout)

这些推特是2015年2月写的，从中可以看到通过知识分享在公司之间建立亲密性的一种发展趋势。Kromhout指出DramaFever在所有技术团队中就采用了无问责事后分析。



DramaFever还鼓励通过代码审查组织学习。除了代码审查，Kromhout描述了快速增长带来的文化挑战，以及需要修复团队之间的相互理解，为此要通过“增进协调，明确我们期望服务如何互操作。这就允许小开发团体能够继续追寻他们自己的想法，同时保持组织整体的标准”。

DramaFever鼓励提高透明度。Kromhout指出，“现在开发人员可以充分访问AWS中的开发环境，我们也支持只读的生产IAM访问”。这种透明度使人们可以直接学习和观察开发环境，这个环境复制了生产环境，而且可以回答实际生产使用中的相关问题。

由于DramaFever大量使用了AWS技术，所以不需要数据中心，相应地不要求员工必须在某个特定环境附近。由于DramaFever面向世界各地的用户，尽管有大约120人的团队主要在费城和纽约工作，但大量员工都分布在其他地方。描述这个环境时，Kromhout说，“我们的员工近的在马里兰，远的甚至在首尔，这并不会对他们的工作有任何干扰”。

为了进一步方便远程工作的员工，DramaFever会议室配有Chromeboxes，这是一个商业视频会议系统，它运行Chrome OS，带一个高分辨率相机以及外接麦克风和音箱。人们可以通过Google Hangouts（环聊）虚拟参会，这样就不必实际到场了。



注意你的组织和团队中工具和工作实践的相互关系：

- 你的团队或组织有怎样的价值观？
- 在实际中你的工具会帮助实现这些价值观还是会带来阻碍？
- 价值观和流程的沟通是否透明？

## DramaFever的工具选择

作为一个小公司，DramaFever有一些预算限制，另外大量的手续流程会带来成本，所以DramaFever在选择工具时很注意，基本上没有采用企业工具，通常都使用开源软件。

Kromhout这样描述这个工具选择过程：“首先考虑期望的功能或结果，然后根据工具满足这些需求的程度做出评价。具体操作的人可能有不同的喜好，这会有一些影响，不过我们还有一组服务标准，所有选择都应当满足这些标准”。

引入新技术时会有一些讨论，包括确定现有的解决方案是否仍然可行，为什么新技术是更好的选择，另外要计算当前员工的额外工作负担。

Kromhout解释到：

在确定开发我们自己的工具还是使用SaaS<sup>注6</sup>时，我们对成本和收益做了评估，这包括构建内部工具相关的成本（因为员工的时间既不是无限的，也不是免费的）。

例如，考虑如何处理日志时，我们计算了当前的日志容量和留存需求，对[在EC2上]用ELK处理日志和将同样数量的日志交给多个第三方来处理的价格做了比较。我们列出了希望如何处理日志。接受报价之后，另外考虑到我们已经花了很多时间维护ELK，我们认为继续使用ELK更合适。

DramaFever努力消除关机时间，甚至希望定期维护时也不关机，并以完成相关工作来消除关机时间的渐进过程度量成功。

Kromhout说：

创建由代码定义的实用基础设施是最重要的事情，因为我们的目标是可以实现所有部件的热插拔，而不能为了维护网站而将网站关闭。这个代码可以是处理JSON的bash代码，通过fabric使用boto定义我们的AWS配置或Chef cookbook或者Python。我们提交所有这些工作的拉取请求，在合并和部署之前要由同伴审查和测试。创建的代码能正常工作时就成功了，我们可以关闭这个GitHub问题，转向看板 workflow 中的下一项工作。



有一点很重要，要记住对你的组织来说“成功”是什么。如果认为一个工具是成功的，一定要知道这意味着什么。在考虑成功的工具选择时，要注意：

- 谁负责做有关工具选择的决策？
- 用什么标准来选择和评估工具及其使用？
- 在工程满意度和客户满意度之间，你更优先哪一个？

当时很多人不敢在他们的生产基础设施中使用像Docker这样的新技术，也有一些人认为Docker没有太大意义。这个案例研究的重点不是专门讨论Docker，而是强调为什么

---

注6： SaaS是软件即服务，应用由一个外部服务提供商托管，通过互联网提供给客户。

这些工程师最终选择了这个工具，他们做了哪些考虑和权衡，以及最终如何把决定落实为所用的工具。

## 分析Etsy

Etsy技术堆栈是一个PHP应用，有大量内部服务（相互依赖性和复杂性很高，不一定是当前越来越常见的更独立的微服务），利用这些服务，可以处理购买、出售、搜索和罗列商品等操作，还可以处理订单的支付。目前有很多知名的大型支付提供商，全世界很多公司都在使用这些提供商的产品，但是Etsy认为需要对这个过程有更多的控制，这促使他们提出了自己的内部支付处理。这就导致了需要PCI合规性以及随之而来的很多问题。主要基础设施分散在位于不同地理位置的众多数据中心。

## 显性和隐性文化

要在一个环境中建立所期望的特定文化，核心是明确地定义一组文化信念和价值观。Etsy从一开始就是一个基于社区的公司，清楚地指出了公司的价值观，具体如下：

- 我们是一个贴心、透明和重视人文的企业。
- 我们会做长远的规划和构建。
- 我们重视所有产品的工艺性。
- 我们相信凡事都不能缺少趣味性。
- 我们会一如既往地保持真实。<sup>注7</sup>

这些价值观很好地鼓舞了员工，使他们紧密地联系在一起，2013年Etsy发展报告中指出员工的关联度为86%，价值一致性达到91%，由此就能反映出这一点。尽管Etsy没有devops团队，没有devops经理，也没有devops工程师，但这些明确而清晰的价值观会在重要实践中体现，并指导devops社区的行为和贡献。Etsy的实践做法包括有同情心、试验和迭代以及鼓励学习型组织。

## 同情心文化

要体现人文，就要展示出同情心。如果你努力让别人生活得更好（尽管这不会改善

---

注7: <https://www.etsy.com/mission>。



你自己的生活），这就体现了同情心。Etsy努力营造一种无问责、适合远程工作的环境，在为员工创建人文工作环境方面做了很大投入。Etsy还鼓励一种“谢谢你”文化，会定期、公开地认可其他人的成就或贡献。通常会认为IT是一个不讨好的工作：做得好时人们不会注意，而出现问题或故障时却会立即发现。对于从事IT工作的人来说，这就不是一个人文环境，所以除了在出现问题时做到无问责，Etsy还积极鼓励在一切正常时说“谢谢你”。

## 感谢的重要性

“谢谢你”或感激文化是建立和改善关系的一个重要部分。认识到其他人做出的贡献，这会帮助人们联结成为更大的团体而不只是作为个人。

研究表明，感谢有很重要的好处，包括：<sup>注8</sup>

改善健康

更强的免疫系统，并降低血压。

提高适应力

能够更好地应对逆境。

增加正面情绪

有更大的幸福感、愉悦感和满足感。

减少负面情绪

减少孤独感和隔离感。

增加协作和合作行为

更多包容和同情。



工具和文化会在组织的整个生命周期中相互影响。注意这些交互可以帮助改善你的文化，还有助于更有效地使用你的工具。考虑以下问题：

- 你希望在员工中间鼓励什么行为？
- 现有工具或工作流的使用情况如何，人们乐于使用还是不愿意使用？
- 如何使用工具来鼓励特定的行为或价值观？

注8： Robert Emmons and Robin Stern, “Gratitude as a Psychotherapeutic Intervention,” *Journal of Clinical Psychology* 69, no. 8 (2013)。

不仅在工程组织中，实际上整个公司都鼓励公共交流，因此，人们想要认可同事的某个工作时，会经常在整个团队范围内发送email或即时消息。如果有人或团队修正了某些bug，增加了特性，对开源做出某些贡献，帮助别人摆脱困境，或者甚至更新了文档，都会受到感谢。运维工程师Jon Cowie指出：

我们的员工目录里有一个按钮，如果你认为某个人很好地展示了公司价值观的某个方面，堪称表率，就可以提名他获得“Etsy最有价值奖”，这就是“谢谢你”文化的一个非常好的例子。这个提名会发送给接收者和他们的经理，这为发送者提供了一个很好的途径，可以通过这种方式对他们欣赏的人或者对他们有过很多帮助的人表示认同。

这种文化可以帮助培养同理心和亲密性，这样人们能更高效地一起工作，使他们以后减少相互之间的责备，为所有员工创建一个更和谐的人文环境。

## 无问责文化

第4章提到过，无问责环境定义为鼓励个人分享故事，并承担提高安全性的责任。John Allspaw是目前Etsy的CTO，他在2012年5月写了名为“无问责事后分析和一种公平文化”的文章，描述了如何转换为采用这种无问责方法处理错误。

失败肯定会发生，这是企业中很正常的一部分，接受这一点是平息情绪的第一步。如果采用传统的方法把失败归咎于个人的错误，然后辞退这个人作为纠正错误的手段，这会制造更多障碍，使人们无法采取具体的行动，或者不能为这个人安排更多培训，Etsy没有这么做，而是采用了一种更系统的方法。Etsy努力平衡安全 and 责任，通过无问责事后分析鼓励大家分享自己的故事。

在用Etsy的Morgue工具创建的事后分析中，会鼓励人们提供以下内容的一个详细说明，包括：

- 动作和事件时间表。
- 观察到的结果。
- 期望。
- 假设。
- 后果和决定。

Etsy鼓励这种行为，提出不会因人们分享自己的故事（错误）做出任何处罚，并授权他们通过描述事件是如何发生的来改善安全性。由于不担心限制知识分享，人们会对自己的行为更加负责，并帮助创建一个更安全的环境，使其他人不会重复同样的错误。

## 支持远程

Etsy的用户遍布世界各地。这意味着他们的应用要全天候（24小时/7）可用。要支持这个应用的人创建一个更好的人文环境，需要分多个时区，使更多的人能够在当地的正常工作时间工作。这有一个额外的好处，这样可以建立一个更大的未来员工库，可以从中聘用员工，而不是只能考虑居住在某些地方的人。

Etsy使用了多种工具来支持这种文化，包括IRC（即时消息或聊天）、email（较长的文本交流）及Vidyo（视频会议和协作）。前面提到过，要把沟通“默认为是远程的”，Etsy具体实践了这种思想，即使人们在同一个办公室里，也会尽可能使用这些工具，而不只是面对面交谈。

由于工具很好地集成到环境和工作流中，在用这些支持远程的工具进行交流时，只有非常小的开销。这有一个极大的好处，可以让远程的员工也在“圈子”里，不仅了解公司做的决定，还能够参与所进行的各种会谈。

这样一来，人们可以在他们认为最舒服的地方工作，但也有少数除外，如数据中心的技术人员，他们必须在数据中心附近。如果员工感觉舒服，就会对公司更满意，如果允许他们采用对他们来说最有效的方式工作，这不仅对他们的健康有好处，还能提高他们的生产力。



除了支持远程工作，组织可以采用多种不同的方式使用工具来改善人们的生活和工作。要谨记以下几点：

- 工具是否会增加或降低员工的舒适度？
- 工具的灵活性如何？可定制程度如何？
- 工具如何影响人们的日常交流？

## 工具加强实践

工具在加强和支持Etsy的实践方面起到了举足轻重的作用。多人一起工作时可能会产生一些误解，Etsy的支持远程文化则意味着建立devops组合和消除这些误解需要更明



确的实践做法。在沟通方面做的额外工作有助于建立一个人文环境，可以支持跨多个时区的全天候轮班待命。为了建立学习型组织并且平等对待所有人，他们采用了一些战略过程来建立沟通。

支持远程工作的员工时，无法利用专门的编码研讨会和非正式的面对面交流来传递信息。这意味着大量交流要以书面的格式完成。人们会在做出决定时、发生改变或出现问题时或者要分享创新时发送email。员工会创建邮件过滤器，以此限制email的数量。他们发现，如果能把所有交流归档而且可以搜索，这会很有用。

Internet Relay Chat(IRC)对于Etsy着力建立的支持远程文化也非常重要。聊天机器人会更新关于部署、警报和配置更改的信息，还能促进系统交互。例如，对于规划要完成的维护或代码审查可以将Nagios告警置为静音。通过相互点赞和推荐，聊天机器人还可以用来促进“谢谢你文化”。

大多数讨论都在公共渠道进行。这些渠道透明而开放，也允许人们加入其他团队的渠道。这些渠道有些与工作相关，有些则与工作不太相关，因为人们并不只是对工作感兴趣。即时聊天有可能打断人们的工作，造成干扰，所以如果有人退出或关闭通知，以便能专心完成某个工作，这并没有问题。

每个Etsy员工都有一个Vidyo客户端，远程员工还有一个网络摄像头和和耳麦。另外会为大屏幕电视和Vidyo硬件留出空间，以尽量减少视频会议中可能产生的摩擦。大多数远程运维团队都会建立这样一个渠道。通过在运维团队工作区域建立这种稳定的Vidyo呼叫环境，远程工作人员如果想听到和看到办公室里发生的情况，或者想要展示些什么时，就能随时呼叫接入。由于能忽略周围的环境噪声和闲聊，这会帮助他们加入感兴趣的讨论，更能感受到自己是团队和文化中的一员。

在Etsy中，撰写文档被认为是一种工作。大多数情况下，使用Atlassian Confluence软件生成的wiki页面都是准确的和最新的。但显然并不包括所有页面，不过Etsy大力鼓励保持文档更新（尤其是值班手册之类的文档）。很多人会花时间做这个工作。尽管有些公司在犹豫是否要写文档，认为文档很快会过时，所以没有用，但Etsy发现，在实践中“尽力而为的wiki更新”对他们很有帮助。

## 购买还是构建

有一种实践做法是总是采用最新最耀眼的新技术，只是因为它们很新而且有意思，但Etsy没有采用这种做法。他们使用的大多数工具都是他们已有的实用工具。可以在前Etsy工程师Dan McKinley的博客文章“Choose Boring Technology”中了解有关这种理

念的更多信息。基本思想是，如果过于关注实现未经证实的新技术，这会耗费时间和精力，而无法全神贯注地对具体的产品特性进行创新，而这才是公司最终的关注点。

Etsy非常强调他们所称的“慷慨精神”，这表示要尽可能回馈社区，可能采用博客文章、大会发言、指导其他员工或者对开源项目（包括他们自己的项目以及其他人的项目）做出贡献等多种形式。他们将作为内部解决方案开发的大多数工具开源，从这个趋势也可以看出这一点。



选择工具时一般要回答以下问题：

- 可以使用我们已经了解而且有经验的某个工具来完成这个工作吗？如果不能，有没有充分的理由？
- 有没有一个现有的工具能够满足我们的需要？如果有，就使用那个工具。
- 有没有一个工具能够满足我们的大部分需要，而且可以扩展或定制？这个工具开源吗？
- 是否需要或者是否有能力、时间和需求来自行编写工具？
- 研究这个问题领域是否必要，是否采用外部工具？

Etsy使用并为社区回馈了大量工具，包括Nagios, Chef, Elasticsearch和Kibana。如果某些工具不能满足需要，就会被替换。Etsy希望监控他们的网络设备，启用一个新设备时能够在最短的时间内进行监控。当时Etsy使用的是Cacti，但是由于它的复杂性，而且要求手动配置，所以后来开发并发布了FITB。

在边界网关协议（Border gateway protocol, BGP）监控、网站监控和综合测试领域，由于这些问题空间的特殊性质，Etsy选择了外部服务或软件。以BGP监控为例，对Etsy来说，这种选择是有道理的，因为BGP监控需要监视所有外部业务流来了解网络路由的影响和问题。他们的网络工程师可以更好地利用时间，而不是重新创建这样一个复杂的监控服务，况且这样的服务已经存在，所以在这种情况下，使用外部工具就很有意义。

## 考虑自动化

有些领域中，手动流程会带来很多问题，过去数年里Etsy做了大量工作来实现这些领域中工作流和流程的自动化。第1章中我们已经看到这样一个例子，手动的部署过程很容易出错，可能需要花费好几个小时，而且非常难回滚，这个过程被一个更优化的

自动化部署工具所替代，即Deployinator。这不是一个一次性的改变，而是一个迭代的过程，Etsy的大多数自动化过程都是如此。

再来看另一个例子，考虑构建新服务器的过程。由于Etsy在自己的数据中心运行，而不是在云中，过去服务器构建是一个手动的过程，从服务器装架到准备进入生产环境需要花数小时甚至几天的时间。第一个自动化是一组很简单的Ruby脚本，可以处理最严重的一些痛点问题，如配置开关和VLAN。接下来几年中，又增加了更多特性，修正了bug，更多痛点问题得到自动化，现在这个工具已经包括一个Web界面，不只是运维团队的成员，任何工程师都可以指定硬件配置和Chef角色，并在几分钟内将一个新服务器投入生产环境。

不过，Etsy工程师不打算只是为了自动化而盲目地将所有一切都自动化。他们知道剩余原则，<sup>注9</sup>这个原则指出，只有非自动化任务留给人类操作人员来完成，这些剩余的任务要么太过复杂，要么太少见，所以不能自动化，或者过于简单或不经济，因而不适合自动化。过度自动化可能导致所谓的“去技术化”（deskilling），即太多任务都自动化实现，以至于人们会忘记如何完成这些任务，一段时间后他们在这些领域的技术就会丧失。



在很多情况下自动化很有好处，对于原本手动完成的重复性任务，通过自动化可以少花时间，并减少错误。不过，它不是万能的。考虑自动化时，要问问自己：

- 你的最大的痛点是什么？
- 什么可以自动化，什么不能自动化？
- 工作流的某些方面是否应该自动化？
- 如果你创建的自动化出了问题，要如何处理？

## 度量成功

为了有意地尝试和鼓励学习，Etsy非常强调透明度和监控。Etsy已经分享了大量工具和流程，由此就能体现其对这一点的充分关注。从监控系统级性能到业务级指标，Etsy努力收集尽可能多的数据点。这些数据对员工透明，即使是对运维没有深入了解的人也能对迭代改进有所认识。这种对透明度和监控的关注不是一夜之间形成的。

---

注9： Tom Limoncelli, “Automation Should Be Like Iron Man, Not Ultron,” ACM Queue, October 31, 2015.



Michael Rembetsy于2008年加入Etsy。他和他的团队要通过Etsy客户论坛上的帖子发现问题，因为那时这个组织还没有真正的监控。频繁的宕机和反应性发现促使Rembetsy和其他领导人要找出一种更可持续的方法来运行这个平台。他们没有试图一次就规划出完备的解决方案，开始时只提出了一个最小可行的监控解决方案，首先从对客户体验影响最大的基本问题开始。

对于具体需要哪些工具，并没有一个清晰的路径，所以他们进行了试验。目标是了解网站、应用和所有相关组件上发生了什么。他们最初选择了Nagios、Cacti和Ganglia，这是因为他们对这些平台比较熟悉，相应的实现速度较快，而且成本低（免费）。

经过频繁的迭代和演进，Etsy采用了一种“度量一切”的实践做法。除了提前规划好想要度量什么，任何人都能很容易地得到所有指标，这些指标可以在事后绘制一个监测图。他们开发并发布了StatsD，这是一个网络守护程序，在Node.js平台上运行，可以监听通过UDP或TCP发送的统计信息，并聚合到某些可插拔的后端服务（如Graphite）。数据每10秒刷新一次，从而支持近实时的数据收集。

这里的共同目标是创建和交付软件。不同的团队根据各自的需要建立监控。没有专门的人负责监控；而是鼓励每个人按其所需或尽其所能做出贡献。关于监控，他们有明确的认识：

- 如果你有问题，就去问别人。
- 如果导致生产问题，运维团队会讨论你需要什么，从而解决这些问题。

作为一个实际devops组合的例子，Daniels描述了与一个不同团队共同完成运维工作的过程，这是一个前端基础设施团队，她的目标是处理发送给她的警报。半夜收到关于一个服务器磁盘空间的警报时（每个系统管理员都会收到这一类警报），她意识到这个团队将日志写入了一个很小的磁盘分区，这个分区比存储大多数日志的标准分区小得多。

意识到这一点之后，她与这个团队讨论关于记录日志的运维最佳实践，指出对于这个问题已经有一些相关文档，并提出两个解决方案作为建议。然后这个前端基础设施团队选择并实现了更适用的解决方案。这种无问责和信息分享的组合正是创建和维持一种“理解文化”的关键。



监控和警报在每个软件环境中都是很重要的部分，在这些领域中使用有效的工具可以带来巨大的好处。要考虑：

- 你的工具如何区分监控和警报？
- 你的工具和流程如何处理不同团队的不同监控需求？
- 你的监控和警报解决方案的灵活性和可定制性如何？

## 动因和决策挑战

从这两个案例研究可以清楚地看出，关于人们日常使用的工具，并不能一夜之间就做出决策。公司时讯、主流媒体和会议室都会展示构成devops工具链的“最佳”工具列表和文章。有些公司推销的解决方案在你的环境中可能很有效，有些公司则有可能阻碍devops的发展，你要如何区分这二者的区别？

工具对于如何完成工作很重要，但是这绝对不是唯一要考虑的问题。并没有“devops即服务”的概念，你无法购买一个能解决所有相关问题的包罗万象的解决方案。要了解重要的一点，尽管工具会影响文化，但是并不能取代文化，所以如果有人试图向你推销一个“定制devops解决方案”，一定要当心，听上去实在太好了，简直不像真的，事实上这也确实不是真的。

除了我们自己的目标和勃勃雄心，还会有其他动因以人际关系的形式出现，如某个组织更倾向于开发商X而不选择其他解决方案，原因可能只是因为开发商X带他们参加了一个赛事或宴会，或者决策人的一个朋友有一家新的创业公司，他想支持这家公司。

也有可能因为工具在某个方面有很好的声誉而选择这些工具，就像人们所说的，“没有人会因为购买IBM产品而被解雇”。要了解你的环境中决策的背后动因，这会帮助你明确如何改进这些过程。

## Sparkle公司有效地使用工具

“我发现你的演示真的很有意思，也很有意义”，George说。“我注意到你在你的笔记本电脑上使用了一个虚拟机，与代码单独管理。你试过使用Test Kitchen创建一个项目来管理你的虚拟机吗？运维团队在测试服务的新实现时就使用了这个工具。这样一来，我们就能复制团队里任何人做的任何工作了”。

“没有，我没有听说过Test Kitchen。我很想看看它是什么，特别是能不能减少启动定制虚拟系统的时间”，Alice说。

“实际上，首先要从ChefDK开始，也就是Chef开发包。发给公司员工的所有笔记本电脑上都已经安装了这个开发包”，George说。“看来我们应当与IT团队协调，确保公司本地开发环境的有关文档已经更新。作为运维工程师，我的新员工文档里就有这个工具。我不知道其他团队没有使用这个工具”。

George展示了快速建立一个Chef cookbook有多容易，其中利用一个预建的Test Kitchen模板镜像了Alice、Geordie和Josie为MongoDB安装完成的定制工作。

“现在，我会把它提交到我们的集中Git环境，你们都可以下拉这个项目来处理”，George说。

Alice做了测试，她克隆了这个项目，并像George那样使用kitchen命令。OS镜像同步后，她的笔记本电脑上快速建立了一个测试环境。

“对MySQL也可以这么做，这样我们就能更快地用真实指标来评估不同解决方案的优劣了”，Josie说。

“不如我们分成两个团队？结对把它插入到我们的Jenkins集群。这样我们就能对软件项目下拉到各个平台进行测试，同时对它们进行评估”，George说。

最后，根据运维成本，所有人都认为继续使用MySQL最合适。可视化指标帮助各个团队看到在这个环境中继续使用MySQL的好处。利用Chef、git和Jenkins，相关的每个人都能分享他们的工作，而不必重复劳动，这使得不同团队的人可以更容易地一起工作。

通过这种协作方法，团队在一周内就为这个评论应用建立了一个初始的演示版，开发团队可以有更多时间与安全团队一起规划他们的骚扰检测算法。由于一直在进行开放的沟通，这让所有人感觉别人听到并考虑了他们的意见，所以这是一个协作得出的决定。

## 小结

一定要明确你的组织的价值观。Etsy公司就有一组非常清晰、很有吸引力的价值观，这会指导他们决定使用哪些工具和技术，还会指导人们在日常工作中如何使用这些工具和技术。



要根据你在团队中观察到的当前活动总结实践。在Etsy和DramaFever可以观察到以下实践：

- 无问责环境。
- 试验和迭代。
- 增量改善。
- 学习型组织。

一旦明确你的当前活动和实践，就能确定你的实践与价值观是否一致。例如，如果你说很重视开源，但是实际上更经常选择非开源的开发商解决方案，或者没有给人们留出时间让他们为开源做贡献，这就是一个信号，说明你的理论价值观和实践价值观并不一致。

要根据你的文化、技术水平和需要来选择工具。你选择的工具可能会随时间改变。即使有共同的文化和价值观，不同的组织或个人也会有不同的技术或业务需要。尽管在这两个案例研究中我们看到很多相同的价值观和实践，但是DramaFever最后选择了一组完全不同的工具。不能说这两个公司哪一个“正确”或“更好”，做决策时你必须知道对于你的组织什么是正确的。

要理解文化和工具有效性的改变不会一蹴而就。Etsy从2008年就一直在完成他们的监控项目，而且随着继续迭代还在继续修改他们的代码。他们为社区贡献了丰富的工具，不过这些解决方案并不是拿来就能解决你的特定问题，尽管它们可能很有帮助。要真正做出改变，这需要一定的时间和不断的实践。

要知道，度量你的进度对于获得成功至关重要。如果你还处于零监控状态，就需要在这个领域多花些时间。可以读一读Jason Dixon的《Monitoring with Graphite》(O'Reilly)，这本书深入地介绍了监控，可以帮助你更充分地利用监控提供的好处。第20章列出了这本书以及很多其他很棒的信息源。

最后，要记住工具不能与有效实现devops的另外3大支柱完全分离。最终工具还是要由人来使用，帮助他们与其他人一同工作，并为人们创建解决方案，所以选择工具时不能不考虑人的方面。工具不仅会影响我们如何工作和交互，也会受其影响，要实现持续的重大改变，就必须考虑到所有这些因素和交互。

# 工具：误区和问题排查

与选择和使用工具有关的不同情况下会出现一些问题，这一章会从更一般的角度介绍这些误区和问题排查。这里不包括特定工具或技术的问题排查，因为这不属于本书的范畴，我们只考虑决策过程和不同工具工作流的相关问题排查。

## 工具误区

对于devops相关工具的选择，很多误区都与特定工具对devops解决方案的重要性有关。

### 我们使用技术X，但所有其他人都使用技术Y。我们必须尽快改成技术Y

第一部分中提到过，devops是一个文化运动。文化包括技术堆栈，而技术的全盘改变（尤其是从管理层强制推行时）会带来成本，使组织整体发展速度减慢。在一个特定技术退役之前，要找出当前环境中已经成为现有文化一部分的工具，了解人们使用这些工具的体验，并观察与其他人的体验有什么相似和不同之处。这种检查和评价有助于明确需要做哪些改变，以及是否需要立即做出改变。

对此有一个例外，这就是升级。技术的升级是必要的。升级拖延得越久，就会带来更多基于可靠性的技术负债，而且需要对兼容的升级路径做更多测试。如果升级过快，起码可以对产品做质量保证测试。而升级过晚，则有可能基于已经退伍的技术来完成升级。



如果只是简单地复制你在成功组织中看到的工具，在你的组织中不一定能够得到同样的结果。要强调过程，而不是强调结果。如果技术X在你的环境中是适用的，那就使用这个技术。

存在某个特定的工具或技术并不意味着你就不能完成一个成功的devops项目。完全可以使用IRC而不是Slack或Hipchat，可以在裸机上运行服务器而不用在某个云上运行，或者也可以有一个整体的PHP应用而不是Go微服务，这些与devops思想并不排斥。devops运动主要关于文化以及人们如何一起工作，如果你能利用一个20年历史的老聊天程序让人们很好的协作，这与使用一个全新的聊天程序是一样的，更重要的是让人们协作，而不是使用哪一个聊天程序。

## 使用了技术X就意味着我们在实现devops

当然，如果有某些类型的工具和技术，对完成devops项目会有很大帮助。之前我们使用了版本控制和基础设施自动化作为主要例子，不过重要的是，要理解为什么这些工具有价值，以及这些价值与完成软件开发工作的人有什么关系。

例如，基础设施自动化允许员工采用一种更优化、更可靠的方式做出变更，可以减少与变更有关的风险和摩擦。

但是如果没有影响到人，技术的影响会大大减弱。如果你开始使用基础设施自动化，但是还维持原来遗留的变更控制过程，这个过程中还存在开发人员的痛点问题，你就不会看到基础设施自动化可以提供的好处。任何一个工具本身都无法修补被破坏的文化。为了有效地使用工具，你需要查看人们如何使用工具，以及为什么使用这些工具，他们打算完成什么工作，以及这些工具对他们的工作有什么帮助或阻碍。

如果只是增加了Chef、Docker、Slack或者讨论devops时常常提到的任何其他工具，并不表示你在“实现devops”，因为对于我们如何一起工作，工具只是其中的一部分。不能说一个工具的存在或不存在会促成或破坏一个devops项目，只能说这些工具对你的文化有什么帮助或阻碍。

## 我们绝对不要选择错误的工具

有些人担心选择“错误的”工具会带来可怕的后果，导致项目或者甚至组织陷入失败。这种担心可能被开发商夸大，声称你“必须”使用他们的产品或解决方案才能做到devops。你可能希望确保你的决定不会产生严重的负面影响，这是很自然的。

这种对特定工具的过度强调是有误导性的。实际上，你应当更多地关注如何在整个组



织中使用工具，以及在使用中可以从正面和负面学到什么经验教训。你不会因为选择了错误的工具而失败，但如果错误地考虑所选择的工具，这才会导致失败。

以基础设施自动化为例，不太可能因为选择了Chef而不是Puppet（或者反之）而对你的基础设施自动化项目的整体成功或失败产生重大影响。肯定有一些小的实现细节或者特定的用例比较适合使用某个工具，从中受益更多，但是从大局来讲，与使用哪一个工具相比，如何使用基础设施自动化工具的影响更大。

如果你了解有关的原则，包括如何适当地使用工具，什么时候自动化是有益的而什么时候没有好处，如何选择新工具并在你的组织里具体使用，你就能做出适当的决策，尽可能地选择合适的工具，而且更重要的是，可以从你的经历中学习经验教训。

新工具和技术在不断涌现，掌握上述原则可以确保你在这个不断变化的大环境下适当调整，所以即使你确实选择了一个最后看起来不算好的工具，也不会浪费太多资源或者一直固守着这个工具。与试图直接选择“最好的”工具相比，能够研究、学习和调整如何使用工具更为重要。

## 可以购买devops成品或devops即服务

devops的影响越来越大，也越来越普及，这导致很多开发商开始在他们的市场营销中增加与devops相关的术语，希望以此赶上这一潮流。有时可能很难区分营销宣传和现实，特别是如果你对这些想法完全陌生，而且被众多以devops为卖点的产品包围时，可能更加没有头绪。

应当保持一种均衡的观点，要考虑4大支柱。除了工具，还要考虑协作、亲密性和规模化。各种工具可以是“成品”或者“作为服务”，但是正如我们在这一章中所展示的，只有最新、最酷的工具还不够，必须有效地使用这些工具才能成功。

devops不只是关于工具。你要如何推销协作即服务？要知道，并不是一个帮助解决特定协作或沟通问题的工具，而是人们一起工作的这种具体行为，你能把它当作产品吗？让不同团队坐下来讨论他们的不同目标、优先级和痛点问题呢？这些都无法作为一个成品或服务来购买。最终，你需要做大量改变，使组织中的人们达成一致，并建立和维护共同的理解（我们把这描述为devops组合），这些改变必须要在你的组织内部完成。

有很多非常好的工具和服务。很多公司提供了能解决实际问题的解决方案。在做出评价时，要谨记4大支柱以及它们之间的交互和交集，问问自己这些公司是否做出了某

种承诺而没有能力交付。（假如你把他们营销手册中的所有“devops”替换成“人们很好地合作”，如果看上去很滑稽，这可能就不是你要买的东西）。

最后，你必须自己完成艰巨的工作，用你自己的人和团队找出哪些工具适用，哪些对你自己的组织不适用。要让整个文化持久地改变，这是买不来的，必须在内部创建。

## 工具问题排查

排查使用技术时出现的问题时，要记住这些是工具，而不是玩具。你应当努力选择帮助你解决问题的工具，而不能只是因为你想尝试或者因为所有其他人都在使用就选择这些工具。

### 我们想找出技术X的最佳实践

找出一组特定问题的最佳实践可能看起来很好，但是这种心态会带来问题。我们选择“最佳”解决方案时，倘若它没有达到我们的期望，我们就会想办法处理这种认知的不一致，而这往往会变成责备。具体解决一个问题之前，可以采用几个关键的策略：

- 明确现在问题的状态。
- 明确哪些必须有，哪些可有可无。
- 找出拥有重要信息的个人和团队，与他们一起进一步限定问题。这里的目标不是找出所有可能性，而是要明确哪些部分需要有弹性，哪些部分是所做工作的重要基础。如果有一个多样化的团队，其中包括分析型、横向型和批评型思维的人，就能在发现潜在问题的过程充分考虑各种可能性（“如果……会怎么样”），而不会停滞不前。

在这个识别过程中，你会发现问题的一个特定模式。一旦得到这个模式，将它与可用选择进行比较。这些选择可行吗？选择其中一个方案会有什么结果？如果创建一个全新的方案又会有什么结果？

要做出决策，为决策过程建立文档，从而说明这是一个理智的决定。你认为哪些弹性因素将来可以改进；哪些基本问题可以支持这些决定。另外还要记住计划是所有工具或技术不可缺少的一部分，尤其是这种技术涉及自动化时。尽管自动化可能有很多好处，但是如果没有适当的计划，最后可能会对一个有问题的过程完成自动化，使情况变得比之前更糟糕。

在完成决策和相应工作的同时，要确保人们花足够多的时间充分考虑和描述“what if?”条件。这样一来，你的团队可以发现一个现在能正常工作的解决方案，得到认可，准备好随着技术的演进和发展而改变，同时为发生不可预见的问题时系统出现故障做好准备。

## 我们无法让人们对一个特定工具达成一致

在较小的组织里，可能希望让每一个人对于使用哪些工具以及去除哪些工具达成一致，而且在足够小的创业公司里，这是有可能的。不过，随着团队和组织规模的增大，这会变得越来越困难，组织规模和复杂性达到一定程度时，及时从使用特定工具的每个人那里得到反馈甚至都不可能，更不用说让所有人都达成一致。

一旦你接受了这一点，即根本不可能达到全体一致，你就会转而寻找适合大多数用例的解决方案。你希望确定哪些人每天都会使用工具，并针对他们的需要和用例进行优化，而不是关注那些只是偶尔使用工具的人。可以召集成立一个测试小组，这个小组由代表这个工具大多数常见用例的人组成。还可以考虑让人们作为beta测试人员，帮助评价可能的解决方案。

很多技术人员可能拒绝改变，你可能发现人们反对引入一个新工具或替换现有的工具，只是因为那是他们熟悉的工具，而不是因为新工具存在某个问题。可以考虑建立一种结构化方式，采用这种方式得到人们对所用工具的反馈，明确他们是否频繁遇到问题，以及这些特定的问题是什么，另外要记住，抱怨声音最大的人不一定代表大多数人的意见。

灵活性固然很重要，但在某些领域，团队或组织中可能要有一些强制性的工具而没有商量余地。例如，如果利用某些工具可以维持SOX、PCI或另外某种合规性，那么在这个领域工作的每一个人都要坚持使用这些工具，这是有道理的。你可能希望这种强制性的工具要尽可能少，不过，肯定有些领域需要有这样一些工具。

## 我们决定采用（或取消）某技术，但是人们拒绝使用（或放弃）

人们在工作中是否愿意使用一个新工具，这在很大程度上可能取决于选择这个特定工具的过程。在你的组织里，如果这是一个自上而下的决定，这个决定可能是一个对给定工具或工作流没有太多经验的经理做出的，他对这些工具或工作流的经验远没有他的下属多，就完全可以理解为什么人们不想使用这个新工具。要研究你打算解决什么问题，另外如前所述，与频繁使用这个工具的人一起工作非常有助于避免这种情况。



要注意如何描述变更以及如何实现。如果员工计算机在某个地点集中管理，可以由IT员工远程推送软件变更，倘若人们早上来发现为他们安装了一个新软件，但是没有任何解释来说明这是什么，更重要的，没有说明为什么要做这个变更，他们很可能会拒绝这个改变（即使对他们来说这个工具最后看来确实是一个很好的解决方案）。

要让人们提前知道什么时候会做出可能对他们有影响的工具变更，尽量让他们有机会参与这个过程。你会发现，有些人在使用一个工具，而你却没有考虑到或者不知道他们在使用这个工具，这些人往往态度很强硬。一旦决定改变，要与他们提前沟通，让他们了解将要做出改变，并尽可能给人们留出时间转换到新工具，或者如果要取消一个现有的工具，应当给人们时间让他们习惯于不再使用这个工具。要解释这个改变考虑到哪些因素，如何做出的选择，并让人们知道他们可以在哪里提供反馈或报告问题。

不论是增加新工具，还是将现有工具退役，这一点都适用，与本书介绍的其他方面一样，沟通同理心非常有助于持续有效地完成改变。

## 规模化

这一点就这个组织态和克服组织在生命周期各个重要时间点遇到的障碍。有人可能很想把这个想法简化为“企业devops”，因为很多人在较小的和创新环境以外考虑devops时都会这么做。不过，经过简化了，尽管确实有特定于企业的问题（这一章会讨论其中的一些问题），不过还需要描述公司如何随时间改变，这会更完整，也更有意义：可能是创业公司的规模扩大，也可能是一个企业组织分解为两个组织。规模化就是组织作为一个实体在整个生命周期中的演变、成长和前进。

## 理解规模化

在一个团队、部门或组织中，并不总能很容易地知道什么时候需要改变，或者需要什么类型的改变。尽管可以提前得到有关这些变化的提示，这可能会有帮助，但总是有限和不完美的。

随着我们在演进过程中的进展时（原来的工作对将来可能有帮助也可能会有阻碍），有意识地根据当前的状态规划、实施以及调整方位会很有帮助。不论这是轻微有限的小动作，还是大幅度的改变。有了这些经验，我们就会了解什么时候需要调整以及如何改变。另外如何应对采用不同策略的不同环境。

在这一章中，我们会考虑组织规模化时需要考虑到的问题。首先，我们会讨论企业devops的前3个支柱（协作、自动化、度量），然后我们会讨论一些其他支柱。最后与前面各章一样，之后还会介绍一些设计原则、模型、框架以及工具等。

# 规模化：拐点

这一章重点介绍检查和克服组织在生命周期的各个重要时间点遇到的障碍。有人可能很想把这个想法简化为“企业devops”，因为很多人在较小的初创环境以外考虑devops时都会这么做。不过，这过于简化了，尽管确实有特定于企业的问题（这一章会讨论其中的一些问题），不过还应当描述公司如何随时间改变，这会更完整，也更有意义，可能是创业公司的规模壮大，也可能是一个企业组织分拆为两个组织。规模化就是组织作为一个整体在整个生命周期中的演变、成长和前进。

## 理解规模化

在一个团队、部门或组织中，并不总能很容易地知道什么时候需要改变，或者要在哪个方向改变。尽管可以提前得到有关这些变化的建议，这可能会有帮助，但是感觉很不直观。

查看我们在演进过程中的进展时（原来的工作对将来可能有帮助也可能会有阻碍），有意识地根据当前的状态规划、实施以及调整方位会很有帮助，不论这是缓慢有限的小动作，还是大幅度的跳变。有了这些经验，我们就会了解什么时候改变方向以及如何改变，另外如何应对采用不同策略的不同环境。

在这一章中，我们会考虑组织规模化时遇到的挑战和问题。我们将深入讨论有效实现devops的前3个支柱（协作、亲密性和工具）与不同类型的规模化问题之间的关系，然后与前面各章一样，之后还会介绍有关规模化的一些误区和常见的问题排查。



# 考虑企业devops

并没有只适用于大公司（有大量员工）的单独的“企业devops”（采用不同的工具和实践）。成功没有唯一的定义，也不存在所有公司和组织都应追寻的唯一目标。组织要适当地发展实力，有必要的敏捷性和均衡性来实现改变。

与小型组织中一样，devops组合也是大公司增强实力、均衡性和敏捷性的核心。一种强调协作和亲密性的文化会加强“弱联系”，使整个公司中有更多的信息流动。在较大的组织中，原则本身并没有不同，只是这些原则的应用或实现有所不同。

有些人有这样的担心，认为devops原则只能应用于小创业公司的绿场项目，而不适合企业组织或者有累积技术和文化负债的遗留系统，但是Puppet Labs 2015年DevOps现状报告的相关研究表明并不是这样：

如果设计时谨记可测试性和可部署性，是可以得到高绩效的。

这个报告的研究人员发现，devops的文化原则适用于任何规模的组织，技术原则（如持续交付和改善部署流程）可以应用于任何设计良好的软件项目，甚至包括在大型机上运行的遗留代码。一个基于微服务的全新项目不会因为它是全新的和有微服务就一定成功，这个项目必须有良好的设计，保证可测试而且要易于部署。这些原则适用于所有软件项目，而不论新旧。

## 采用devops战略扩缩组织

成功的组织必须知道如何规模化，也就是说，如何根据需要扩展或收缩。取决于具体上下文，规模化对于不同的人可能有不同的含义。在这方面，规模化也是大众模式的一个例子，为了有效地在你的组织内部或者与其他人讨论规模化，首先需要明确讨论的是哪种类型的规模化。例如，规模化可能是指：

- 扩大客户群体。
- 提高收入。
- 扩展一个项目或团队来满足需要。
- 维持或改善人与系统或与所花费资金的比率。
- 比竞争对手发展更快。

另外，可能还会用一些限定词描述所做的工作，这让情况更加混乱。例如，“大规模”系统。利用托管服务，一个工程师在几分钟内就可以快速部署和撤销成百上千个系统，而不再需要花费几个月的时间，在这种情况下，“大”是指什么？如果从快速发展的系统可用性角度来考虑，有没有只适用于组织中某个子集的一组原则、实践和技术？

简单的说，答案是没有。我们在第四部分中介绍了Etsy的案例，我们指出，2015年Etsy有大约800名员工，150万活跃卖家，2260万活跃买家，年毛销售额达到19.3亿美元。来看另一个例子，2015年Target有大约347000名员工，年收入为720亿美元。除了规模上的差别，这两个公司都同样基于它们的当前文化采用相应的原则和实践来选择对它们最适用的发展路线。

## 精细规模化化的问题

更进一步，如果试图找到一种能解决一切问题的机制，尤其是想要把人从系统中消除，这是完全错误的。在真实世界里，建筑师设计建筑时，他们会基于多年的教育和经验、直觉以及机械过程来完成他们的设计。建筑、周围环境、地区、历史和环境研究等方面的需求都会影响这个设计。

利用草图和3D模型可以最终构建一个能够看到和体验的真实构造。当前文化会影响建筑师如何想象和设计建筑及其内部空间。<sup>注1</sup>对于一个建筑，没有任何一个设计可以解决所有情况下所有可能的需求，我们的软件项目和组织也是如此。

如果根据系统的单个部分和我们自己的经验对系统做一些假设，这会很危险。构建、管理和使用系统时，系统本身会产生一组丰富、复杂的响应。复杂的系统不会只有简单、线性的故障，导致故障的根本原因可能不只一个。设计和规模化系统时需要考虑很多因素，尤其是它们之间的交互。

举例来说，一个数据库服务器读50次相同的数据与读50次不同数据的响应方式就不同，这取决于所用的软件以及软件如何配置。由于现在数据库是分布的，不再是只在一个服务器上，所以特性和行为也会改变。不要假设可以由过去的行为预测将来的行为。

管理这个数据库服务器的经历会让人们积累知识和经验，他们可以用这些知识更有效地“武装”其他人，这有助于发展团队，而不再只有少数几个人掌握这些知识。另

注1： Jun'ichirō Tanizaki, In Praise Of Shadows (New Haven, CT: Leete's Island Books, 1977)。

外，要以一种一致、可重复的方式简化系统的管理，这对于减少复杂性至关重要。简化过程本身并不简单。也许某种方法在一个环境中可行，但在另一个环境中可能完全是错误的。

## 规划规模化

要明确你的系统要有怎样的表现，以及目前总的来讲什么更重要，从而能够针对当前环境构建一组优先的系统。要了解你的目标，这很关键。这是否只是学习过程中的一个练习？是不是在响应系统故障？是否在恢复一个安全问题并重建信任？

对于软件，人们会讨论设计工艺，这个领域取得了很大进展，甚至因此出现了“架构师”这个头衔。有关软件架构的种种故事会影响我们的选择，通常会阻止我们以全新的方式查看软件。并不是整体的软件结构不好或者不如微服务。通过分析技术、流程和冲突策略，可以确保我们对环境中的灵活性和不灵活性做出理性的决定。要有意识地设计组织中的弹性环节，将来能通过静态或动态响应更精细地实现改变，因为我们知道我们有一个强大的基础，而且有适当的灵活性。

## 组织结构

重构团队组织方式可能有利于规模化。跨职能的小团队中，成员有多种不同的技能（例如，前端和后端开发、设计、UX和运维），他们共同完成一个项目或产品，具备从头构建产品所需的所有技能，从而可以覆盖更宽的领域。

不过，单职能团队也有好处，如可以更大程度地实现知识分享以及团队或部门的专业化。如果你发现单职能团队与组织中的其他部分仍然能很好地沟通和协作，就不要只是为了重组而重组。不论整体团队结构如何，都要有跨团队的沟通，这对于整体实现一个有效的组织至关重要。

层级组织可能会抑制创新，让人感觉没有发言权。不过，权力变化和状态差别也能促进和改善有效性。如果组织尝试通过大规模重组来消除层级结构，导致组织过于平面化，这可能也会带来问题。一个组织要有适当的复杂性，一方面要对组织有利，另一方面不能打击员工的士气，要在这二者之间努力达到平衡。

## 分布位置

由于组织或团队分布在多个不同的位置，这就需要特别强调公司整体的沟通技能是否有效。



如果公司分布在多个位置，很快你就会明显地看到，沟通和决策过程很大程度上依赖于人们能够进行面对面交流。如果你有短期或中期规划希望扩大公司的地域分布，就要确保每个人有足够的书面沟通能力，这会有很大帮助。

从组织角度来看，分布在多个位置会带来与IT和基础设施有关的一些新问题。如果只有一个办公室或一个场地能够访问某个很重要的资源（可能是一台现代打印机、一个专门的技术支持团队、一个更快的互联网连接或者另外某个资源），在其他地方工作的人就会感觉自己二等公民，这种情况很可能会伤害员工的士气和整体生产力。要确保不同地方的基础设施都能处理正常的工作负荷和工作模式，这也很关键。

要在全球层次上理解客户的文化差异和期望，从而了解如何配备人员提供本地支持。由于技术飞速变化以及全球竞争的加剧，要让你的公司在竞争中脱颖而出，这可能更需要更多样化的人员分布。



为了让一个公司发展壮大，突破某个瓶颈点，你必须明确如何让分布在不同地方的团队很好地工作。如果做不到这一点，你就不能对影响组织的改变快速做出响应，而且会导致重复工作，另外由于人们要努力弥合这些距离，这也会降低个人和团队的满意度。

## 团队灵活性

关于有效的团队规模已经有大量研究。太小的团队通常缺乏完成所有工作所需的资源，比如人们的时间或知识。更大的团队，尤其是超过9到10个人的团队，由于有太多的人际交互和往来关系，要做出及时有效的决策可能比较困难。大团队还可能容易陷入一种群体思维，为了整个团队的一致性而不允许个人有不同的观点，这会削弱他们的创造力和问题解决能力。

团队规模要维护在5到7个人，而不是无限制地扩大团队，这意味着如何再聘用人员就需要创建另外的团队。更多的团队意味着需要更多管理者和领导。从事管理不是职业提升，而只是一个职业改变。内部提拔对于保持文化至关重要，要提拔那些能干的人，而且他们要对强调人的职业感兴趣。

## 太多层级管理？

对于较大的公司，常有这样一种抱怨，认为有太多层级管理，导致事情很难办。层级体制定义为管理大组织的一种管理体制，通常因为太过复杂、效率低下或不灵活而饱受诟病。正是由于这些因素，尤其是不灵活性，导致一些人认为devops项目在较大的组织中无法成功。过去数年里，在这个领域对于消除不必要的层级管理进行了大量研究。

有些组织将这一点做到了极致，最突出的是Zappos向合弄制的转变，所有决定都通过自组织的团队做出，而不再通过一个传统的管理层次结构。引入合弄制时，鼓励人们接受这个改变，也可以选择拿着丰厚的离职补偿离开。

Zappos宣布转变到合弄制之后的两年间，18%的员工选择离开。如果是采用一般的层级管理体制的组织，很少会鼓励员工离开，所以不确定是否是这些离职补偿让那些离开的人追寻不同的理想或目标。

很多Zappos员工指出，在合弄制之下，职业发展不太清晰，而且权力真空最后可能会非正式地被没有经过管理培训或没有管理经验的人所填充。缺乏明确的权力结构并不表示权力差别不存在。这些差别仍然存在，只不过是隐式的，而且没有正规的过程来解决。

与实现合弄制相比，可能还有更好的方法来应对不必要的繁文缛节。如Max Weber所述，层级体制的发展对应于之前的君主制和独裁制等管理体制，即一个人几乎完全控制所有其他人的想法，这些人几乎没有办法检查和权衡。在某种程度上，Weber认为层级体制是组织和控制人类活动的最高效和最合理的方法。最近人们对基于价值观的领导力进行了研究，希望一方面获得层级体制的好处，另一方面没有太多不必要的层级管理影响工作的顺利开展。

## 组织生命周期

我们可以从两个主要角度考察一个组织的生命周期：

- 内部和外部压力。
- 组织的成长和衰落。

在组织生命周期中存在大量的多样性，因为新的商业模式和投资方法为公司改变、发展和寻求成功提供了途径。

成长阶段的内部压力表现为组织自然地成长，要招聘员工来提供更多产品，开发更多特性，工作效率更高，以及为更多客户提供服务。可能着眼于未来的组织成长，采用先发制人的招聘方式，或者由于意识到当前的人员被过度利用，作为对应对而希望招聘新员工。

在衰落阶段，当公司意识到业绩不如预期，主动减小规模或合并时，就可能出现内部压力。这种改变是否有效会对其未来前景有很大影响。

衰落阶段的外部压力可能来自于全国或全球经济状况、竞争优势的改变或者一个公司由于某些原因（如其产品或专利）被收购或被拆分出售给其他组织。同样的，组织对这些事件做出反应的速度会影响它将来的表现，以及是否能够从中恢复。

## 消除“抢占资源”和“僵尸”项目

在一个组织的生命周期中，要考虑当前项目是否仍在为组织增加价值，这很重要。不论是成长还是衰落阶段，都要找出“抢占资源”和“僵尸”项目，这有助于组织成功地实现改变。“抢占资源”和“僵尸”项目会阻碍组织的成长，或者加速组织的衰落。不论哪一种情况，组织改变时都很适合对这些项目进行清理。

“僵尸”项目是那些占用时间和资源的项目。每个人都知道他们的项目日薄西山，但是没有人能停止这些项目，有时是出于职位安全的考虑，或者是因为停止这些项目对人们会有影响。

“抢占资源”项目是从其他项目攫取资源和能量的项目。通常“抢占资源”项目很难识别，而且由于人们的信念，这些项目很难消灭。有时“抢占资源”项目是因为一直以来的技术债务产生的，有时则是因为信息匮乏而创建这些项目。



处理“抢占资源”或“僵尸”项目时，首先要做的是与所有受影响的人交流，让人们能更好地了解情况。有些人可能拒绝改变而导致项目继续拖延下去，很多情况下，这种交流可以帮助平息这些人的情绪。一般来讲，人们都希望参加有意义的项目。继续做一个要停止的项目是没有意义的。

“抢占资源”和“僵尸”项目都可能非常困难，因为可能有一些核心人员全身心地投入这样一个项目。他们可能甚至没有意识到这个项目对于公司整体来说完全只是耗费



人力物力。对于这个项目，他们可能无法面对现实，别人说项目不行时，他们可能会认为受到了个人攻击。这些人投入了如此多的时间和精力，要劝说他们放弃“这个宝贵的”项目会很有难度，但是这很有意义。如果人们原本对项目充满热情，这种热情一直都有。你不需要再培养他们的这种热情，只需要将其引导到对公司有意义的其他方面。

## 发布周期的影响

有些组织希望加快其发布周期，通常会从一种瀑布过程（变更要花几周或数月时间）转为更小、更频繁的发布。完成变更的速度越快，团队就能更快地对外部和内部压力做出响应，如更快地修正所发现的bug和问题。

不过，有些领域中，一味地注重快速发布可能并没有太大意义。要考虑两个主要问题：

- 一般发布软件的难易程度如何？
- 发布版本的重要性如何？

如今，随着互联网的普及，并不是开发的每一个软件都要设计为全天候可用，或者要持续地更新内容。要了解和权衡项目发布的重要性和复杂性，明确最适合各个项目的发布周期。整个组织中不同的项目可以有不同的发布节奏，这样可能会更好。

大多数情况下，移动应用会受各自移动平台（如Google Play、Apple的App Store或其他平台）发布过程的限制。每个应用商店和平台都有自己的规则、限制和时间表，所以不太可能太频繁地更新（如每周一次）（另外，要考虑到，用户有时候会认为太频繁的更新更多的是一种打扰而不是改进，因而不愿意更新移动应用，特别是每次更新都要求他们注销时）。

嵌入式软件的发布甚至更复杂，更耗费时间。例如，汽车内置软件通常无法很容易地更新，所以倘若存在严重问题，就可能需要大范围、成本高昂的召回，而且很不方便。电视或微波炉等设备的内置软件在安全方面可能没有太大问题，但同样一旦发布就不容易改变了。随着越来越多设备都有了内置的网络连接，向嵌入式软件发布更新更有可能，但是可以空中下载软件升级的设备存在很多要考虑和解决的安全问题。

还必须考虑软件对软件使用者的生活可能带来的影响。要注意这一点，这会帮助你规划发布周期以及工作的其他方面，如根据项目的重要性计划项目的维护窗口或岗位轮换。

与银行网站关闭相比，如果一个社交网站关闭（可能由于一个计划外的系统故障，或者因为要完成计划内的维护），情况可能没有那么紧急，不过即使如此，人们无法访问Facebook时肯定也会拨打紧急服务电话！

如果一个bug错误地告诉用户他在推特上的关注者为0，与一个投资网站上的bug错误地告诉客户他们的投资和退休帐户余额为0相比，前者的影响要小得多。

尽管个人信息的泄漏绝不是小事，不过如果社会安全号、信用卡数据或健康记录被窃取，情况会严重得多。

你能多快完成更改、需要多快完成更改以及这些更改中再次出现的错误都会影响你的选择。

看起来非Web软件可能更容易处理，这些软件不会太快更新，但是这也意味着更难修正所出现的bug。在客户或产品方面越规模化，出现故障或问题的范围就越大。

与一个私有企业相比，上市公司现在必须以一种更可持续的方式考虑他们的选择对利益相关人的影响。上市公司可能必须遵循额外的法规和限制，如某国的萨班斯-奥克斯利法案（Sarbanes-Oxley, SOX）。SOX合规性要求对涉及金融方面重要数据的有关方面有额外的控制，这会影响如何开发和部署与这些数据交互的代码。

## 复杂性和变更

一个组织成长的规模、复杂性或拐点会以多种方式影响其实现devops。组织更庞大或更复杂，他们就必须绕过（或解决）更多的现有约束，大型企业环境和公共事业部门都是如此。对于这些组织，长期存在的层级体制会限制团队之间以及政府中组织之间的协作和亲密性。

政府还有更严格的法规，在采用破坏性技术和实践时必须考虑这些法规。在一个组织中，违反规则可能有不好的结果，其负面后果取决于这个结果；而如果违反法律，不论有什么结果，后果都很严重。

另外，对于政府组织，合同和激励机制可能进一步导致开发、运维和其他重要团队的孤岛化。如果团队对其他团队的成功没有兴趣，协作和合作尤其有难度。

不过，就像企业不要过于关注结果一样，如果能帮助政府组织找到一种方法，限制其过多地关注风险，这会很有意义。在后面的故事中，我们将分享如何减少交付更改的

时间和纳税人的成本，同时明确糟糕的生产发布可能导致的问题以及团队之间协作遇到的挑战。

## 团队规模化

基于共同的工作、目标、相互依赖以及实现成功的责任感，团队会有效地协作。这一节我们将讨论在组织的生命周期中对团队成功最有帮助的各个因素。



有些组织结构会把人们锁定为某些角色，或者人们总有一种恐惧心理，这可能导致人们只关注优化“我”的工作，而不是“我们”。选择有利于某个人的流程和工具可能会带来短期收益，但是从长远来看，这对于团队和组织来说可能是不可持续的。

在我看来，工作最出色的领导人从不说“我”。而且这不是因为他们训练自己不說“我”，而是他们不会只考虑“我”。他们想的是“我们”；他们考虑的是“团队”。他们了解他们的任务是让团队发挥作用，会承担责任而不是避之不及，而“我们”会得到好处。因为这会建立信任，而信任会促使你完成任务。

——Peter F. Drucker

在大型组织里，这些因素有时会划分到特定的角色，尽管作为个人我们的技能和兴趣可能有重叠。这会带来问题，因为没有建立必要知识的时间。更糟糕的是，可能会形成一种声望阶梯，认为一个角色比另一个角色更重要或更好，这会阻碍信息的流通。

那么，大型组织中如何传播决策的有关信息，使人们对自己的工作有归属感或满足感呢？

## 发展团队：规模化时的招聘

规模化团队时的一个关键要素是要发展团队。组织必须考虑整个生命周期中的招聘，尤其是成长阶段。在这一节中，我们将讨论在devops环境中有效发展团队的有关问题。



要指出重要的一点，尽管这一节要讨论devops环境中招聘和保留团队的有关方面，但这不是一个教你招聘神秘的“10倍devops工程师”的指南。应该记得，第13章中曾经说过，devops并不是一个职位头衔。

不能只聘用了解基础设施自动化、云或者容器的人，组织和团队需要注意评估他们的特定需要，另外要解决招聘中的人际关系和文化方面，这对于创建和维护devops文化很关键。



发展团队时经常出现的一个问题是员工的培训成本，这包括让初级人员或刚毕业的大学生提升层次，从而能独立开展工作，还包括为老员工持续提供支持并提供发展机会。如果没有投入时间和资金为员工提供培训和发展机会，初级成员可能只能做底层工作（团队中其他人都没有时间也不愿意去做的工作），而无法提升。

如果只是将IT等领域看作是成本中心，而不是价值贡献者，这样的组织不会为招聘建立适当的预算，会认为这是不必要的。“这个工作应该自动化完成，而不是招聘专门的人员来完成”，他们可能会这么说，尽管自动化肯定是有意义的，但是并不能完全取代人。不是所有工作都能自动化或者应当自动化，而且通常情况下，自动化越复杂，出现问题时就需要更多的人为干预来维护和排查问题。正如第四部分中讨论的，没有一个自动化技术可以取代开发中人的部分，而且不应将自动化看作是一个减少成本的措施。

人们通常不愿意聘用更初级的人员，因为他们担心让这些初级人员完成“实际”工作要花太多时间，或者需要更高级的团队成員花大量时间来培训和指导这些新人。不过，如果不愿意在培训和发展初级人员方面有所投入，很可能导致团队更单一化，而且这会成为一个不太支持成长和发展的环境。另外，要谨记Brooks定律：“投入更多的人到一项延迟的工作上，会导致该项工作更加延迟。”这个想法是软件工程师Fred Brooks在1975年写的《人月神话》一书中提出的，Brooks自己承认这过于简化，<sup>注2</sup>只是累加了为团队增加人员所需考虑的成本和开销。

新团队成员能够真正有效工作之前会有一个上手时间，甚至最有经验的工程师或高级工程师也要花一些时间来熟悉新项目和新的代码基。另外现有的团队成员必须花一些时间来帮助新人快速进入状况，所以这些时间就不能用来完成他们本来可以完成的其他工作。随着团队规模的增大，沟通开销会快速增加，不是每一个任务都能很容易地划分为多个人可以分开完成的工作。想要为一个已有的项目增加更多人手时，一定要谨记这些约束条件，要考虑额外增加人手是否有必要，另外是否有好处，这一点很重要。

## 分包工作

即使在较大的组织中，分包工作（通常称为外包）也是一个要考虑的选择。传统上，希望降低成本的组织往往会把IT或运维等被看作是成本中心的领域作为外包的主要目标。

---

注2： Fred Brooks, *The Mythical Man-Month* (Boston: Addison-Wesley, 1975).

要记住最重要的一点：分包工作尽管在预算上可能会带来一些节省，但是由于个人和团队之间的协作和亲密性减少，这往往会带来成本的增加。将一些团队或部门外包，而另外一些仍留在组织里，这是导致组织中出现冲突的一个主要来源，可能是直接冲突，也可能是间接冲突。下面是冲突的一些表现方式，以及处理这些冲突的一些解决方案：

### 外包会建立职能孤岛

孤岛最大的问题之一是它们之间缺乏沟通和协作，人们往往会独占知识和信息，同时把责任推给其他团队。一个团队被外包时，通常他们可能会感觉由于他们是“外包者”，没有人愿意与他们分享信息，或者他们无法掌握最新动态，所以要确保内部团队与外包团队之间能够进行顺畅的双向沟通。可以使用共享的沟通媒介（如覆盖这两个团队的群聊或邮件列表），或者鼓励定期状态更新，这些都能帮助分享信息。

### 外包团队地位“更低”

在工作的社会层次结构中，不论是正式还是非正式，外包团队通常会感觉他们比内部团队地位低。在组织层次上，可以让外包团队参加团队庆祝等活动，这会有帮助，因为如果人们感觉得到认可，认为自己也是团体中的一部分，他们在工作会更更有动力。在个人和团队层次上，要警惕任何对外包员工区别对待的人，不能容忍这种行为。

### 内部和外包团队之间的责任冲突

内部团队可能总把好工作留给自己，而把枯燥或繁琐的工作留给外包团队，或者试图把责任（通常是责备）推卸给外包团队，不论怎样，都会因为团队之间的责任产生冲突。正式而明确地规定责任有助于消除这方面的混乱或紧张气氛（另外要经常沟通），而且要尽可能想办法让内部和外包人员分享职责或项目，这也能帮助建立一个更协作的环境。

如果组织的态度不正确，认为外包只是一种降低成本的措施，个人和团队也无法改变组织的这种错误态度，尽管如此，关注这些方面可以使他们与外包团队尽可能有效和协作地工作。



每个人都有自己的优点和缺点。与其他人有关联时，这些优缺点可能会变得更好或更糟。评价一个人时，要在整个团队的上下文中考察，这很重要。有时一个人可能不适合某个团队，但是对整个组织很有益。

这里有一个需要平衡的有趣问题：一方面要确保团队继续成长和演进，另一方面还要保持舒适平和的氛围，每天一起共事的人不要变动太大。随着组织的成长和衰落，变动是必要的。在一个小的创业公司里，每个人几乎都相互认识，人际关系很紧密。而当组织从50人扩展到100人甚至更多时，我们与组织中其他人之间的联系就会变弱。

## 保留

在竞争激烈的技术行业，保留员工对于公司老板越来越重要。员工保留不仅会影响团队生产力，还会影响士气。如果同事频繁离开，这会对留下的员工带来额外的压力，同时也暗示团队或公司存在更大的问题。如果很多员工离开的原因是一个新职位薪水更高或者因为他们对公司的发展前景表示担忧，这对于留下的员工通常也不是好事。

尽管员工离开的一些原因不是可控的（如由于家庭原因，如果当前职位不允许远程工作，他们只能选择离开），不过确实有一些因素可以帮助保留员工。这一节将分析这些因素，如果组织希望保留他们花了很大功夫招聘来的员工，可以考虑这些因素。

## 报酬

钱不是一切，越来越多的人会选择健康的工作环境和他们认为合适的公司，而不是更高的薪水。尽管如此，人们还是希望得到有竞争力的薪水。最近的一项研究表明，对于在一家公司工作超过2年的员工，从他们的整个职业生涯来看，最后挣的钱会少得多，10年就会少50%。<sup>注3</sup>这里有一个至理名言（特别是在个人贡献者之间广为流传）：要想大幅度提高薪水，最好的办法就是换工作，与新公司协商一个更高的起始工资。平均来讲，留在公司的员工只能得到大约3%的工资涨幅，如果考虑到2%的通货膨胀，实际上只比1%稍多一点。不过，如果换工作，他们可以得到10%到30%的涨幅。即使你很喜欢当前的这家公司，但一段时间后也很难漠视这种差异。

为了应对这一点，首先要确保一开始付的工资就有竞争力。通常老板喜欢给最低的起始工资作为底线。这往往会使少数群体的成员受到不公正的影响，他们可能比男性白人的薪水少得多。如果能提供达到行业平均线的工资和福利，会有助于吸引这些人应聘，其中很多人之前职位的薪水可能过低。薪酬协商过程、薪酬级别（如果你的公司采用）以及其他与报酬相关的问题要保持透明度，这会帮助保留员工。人们不仅希望得到有竞争力的报酬，而且还要得到公平的对待。

---

注3：Cameron Keng, “Employees Who Stay in Companies Longer Than Two Years Get Paid 50% Less,” Forbes, June 22, 2014.



有一点会很有帮助，就是加薪过程的透明度，特别是有些员工可能因为协商薪酬受到处罚而不是奖励，要保留这些员工，加薪过程的透明度非常重要。这个过程要明确而且（在公司内部）公开，这是保证透明度的关键。与有明确指标的定期加薪过程相比，如果加薪需要人们主动要求，或者只有当管理者考虑的时候才会加薪，这更容易导致无意识的不公正现象。如果有明确的薪酬级别，而不只是依赖管理者的评判，也能减少这种不公正。要确保每个人（包括管理者和个人贡献者）都了解加薪过程（如果你的公司有年终分红，这还包括分红过程），另外知道如果他们对这个过程有疑问要找谁去谈。

## 非货币福利

公平地提供有竞争力的报酬很重要，但是一旦员工得到足够好的报酬，能够保证很好的生活，而且有积蓄而不用担心房租大幅上涨（尤其是像纽约和旧金山这样的租房市场），非货币报酬通常会比额外的加薪更有意义。对于较小或不太成熟的公司，可能无法像资金雄厚或利润丰厚的大公司那样给员工很高的薪水，这种补贴就会是一种吸引和保留人才的好办法，尤其是在这些租金昂贵的地区，可以以此吸引那些有实力但付不起房租的人才。

需要指出重要的一点，我们谈到补贴时，并不是指像办公室里放啤酒的冰箱和乒乓球桌之类的东西。这种“补贴”往往会营造一种气氛，使公司更像是一个兄弟会而不是一个专业的办公场所，而且有些人在这种环境下会感觉不舒服，对这些人来说这种补贴没有任何好处，比如一些女性、不喝酒的人或者只是不希望在工作时玩桌球的人。为员工提供伙食可能是一个加分项，尤其是根据各种饮食限制提供了多种选择的健康伙食，但是要避免提供早晚餐（尤其是晚餐）作为补贴，因为这往往暗示了一种文化，希望人们经常早到和工作到很晚。

可以考虑的福利包括：

### 远程工作机会

不论是为了吸引更多的应聘者，还是保留要搬离公司某个办公地点的员工（例如，他们要搬走与孩子们在一起，或者要离父母更近，也可能想搬到生活成本更低的地方），如果允许他们远程工作，这可能是一个很大的福利。

### 教育机会

这有不同的形式，可能是请讲师到现场讲授，或者派员工参加会议或培训研讨会，使他们能学习新技能或者提高现有的技能，也可以为希望在职业相关领域继

续接受教育的人提供学费或报销书费。对人们来说，个人以及职业发展很重要，所以为他们提供条件（以及时间）来获得这些教育机会可能是一个很大的福利。

### 灵活的工作时间

除非有充分的理由要求人们在特定的时间工作，否则有一定的灵活性会有很大好处。类似于远程工作，这会体现出对员工和团队的信任，以及尊重他们在工作之外的生活和责任。灵活的工作时间允许人们有自己的爱好，避免上下班高峰时的通勤，或者可以在照顾家庭的同时在他们以及团队其他人方便的时候完成工作。

### 工作与生活的平衡

每周连续工作50~80小时对整体生产力会产生负面影响，而不是正面影响。要确保允许和鼓励员工在合理的时间进入和离开办公室，而且不要让他们在家里还要工作或不断检查email。为此最好的办法之一就是管理层做好表率，如果员工看到他们的主管在晚上10点或早上5点给他们发email，要那么晚或那么早回复email会让他们很有压力，不论这种压力是有意的还是无意的。如果员工要值班，每次换班可以为他们提供额外的休息时间，这也是一个很好的福利。

### 带薪休假

要确保你的公司提供休假时间，而且人们确实会利用这个休假时间。在国家法定节假日之外，要避免对休假时间的使用做任何强制安排。例如，如果可以休假10天，但是要求其中8天必须放在圣诞节到元旦的那两周，一年中另外50周里只剩下2天可以休假，这绝对不是人们想要的（特别是考虑到并不是每个人都庆祝圣诞节）。有些公司考虑采用无限制休假政策，但Mathias Meyer（TravisCI持续集成公司的CEO）的一篇文章解释称这也可能有问题。

### 退休计划

组织起码要为员工提供参加一个退休计划（如IRA或401(k)）的机会。另外，要考虑按员工贡献确定相应的百分比。如今越来越少的员工参加退休计划，如果公司能承担加入退休计划的费用，这可能是一个很有意义也很有竞争力的福利。如果做不到这一点，可以考虑采取其他措施，如每年请一位财务顾问到公司为员工提供免费咨询。

### 医疗保险

尽管大多数全职员工的医疗保险都是标准的，但医疗保险计划的覆盖面可能有很大差别，尤其是在美国。更有竞争力的福利包括提供多种计划让员工从中选择、家属承保等。如果组织支付每月的全部保费而不是从员工的工资中扣除，或

者可以访问一个在线健康门户服务来回答员工的医疗和保险相关的问题，这也是一个福利。

### 随意着装要求

如果老板放松办公室着装要求，可以减少员工的长期开销而不会产生成本。尽管职业装看起来可能更专业，但是有些任务要求员工在户外工作，或者需要体力劳动，职业装就不如休闲装合适。对有些人来说，能够自由地选择着装可以反映他们的自主性。在一个有着严格层次结构的环境中，这是在组织中做出改变的一种方法。不同层级的员工更随意地着装时，他们相互之间更容易交流，这会增强整个组织的关系。

### 交通福利

交通福利可能包括预付交通卡、公司班车、自行车锁，甚至停车服务。通过提供相应的机制来减少员工上班的交通或停车拥堵，老板可以缓解导致员工相互竞争的紧张状态，同时可以降低员工的通勤成本。

另外，在某国还有一些关于交通福利的条款，可以有一定程度的免税。这些福利包括交通卡、拼车和泊车等项目。可以查看你所在地区的相关法规。

### 中立的设施

提供中立的厕所可以为人们创建一种更包容的环境，对需要帮助的人也能提供更大的方便。当地法规在厕所和合法标识方面可能有不同的规定，所以不是所有地方或者每一个厕所都能做到这一点。不过，只要有可能，如果能提供中立的厕所（以及更衣室或浴室），会让更多的员工感受到舒适和安全。

### 现场日托

提供现场日托可以提高整体的工作满意度，并吸引有小孩的员工。由于有孩子的员工可以减少因为看护孩子等问题导致的请假旷工，那些没孩子的员工不再担心要为这些照看孩子的父母们代班。

总的来说，在货币或非货币福利方面，员工不应该感觉公司对他们有欺骗。如果人们感觉得到了公平的对待和照顾，而且有解决问题或抱怨的途径，这些员工往往很幸福。

### 成长机会

除了薪水和工作-生活的平衡，人们离职的最主要的原因之一是缺少提升的机会。<sup>注4</sup>

注4：Katie Taylor, “Why Do People Actually Quit Their Jobs?” Entrepreneur, July 16, 2014.



没有人希望找一个没前途的工作。人们希望有机会发展自己的技能，并展示出这种成长，这种成长可能表示更多的独立性或者更多可参与的项目，也可能是得到信任来完成更大的项目，或者有成为领导的机会。

要记住，领导并不只表示管理。在一些公司里，只有管理线有明确的职位等级，这也是员工职业提升的唯一道路，但是很多技术领导人没有必要进入管理层。对于这些人来说，领导可能是指领导项目以及扩大其贡献对组织的影响。要确保非管理者也有进步的途径。理想情况下，这可能意味着除了一个管理线，还要建立一个技术线，明确指定个人贡献者成长的等级。

要有一个明确的成长和晋升流程，这对于管理人员和个人贡献者都很重要。要确保员工了解不同职位等级的定义，有足够的细节可以让人们清楚地看到他们要从一个等级上升到下一个等级需要做什么。晋升流程应当明确，而且是公开的。如果这个“流程”中只包括管理层暗中点名的人，未选中的员工就会很受打击，而且往往会滋生无意识偏见。所有员工都应该有成长的机会，而不能只是老板或CTO的朋友才有机会。

除了这一点，要确保员工有机会研究公司里其他感兴趣的领域。例如，如果一个软件开发人员在多年的开发工作后想要研究运维或安全领域的某个问题，倘若当前公司没有为此提供任何途径，他们很可能会去另一家公司开展研究。尽管管理者通常不应该从其他团队“挖墙角”，但他们应当考虑到人们的兴趣和职业目标可能会改变，并尽可能妥善处理，有些公司会完成“高级岗位轮换”，即员工一旦在公司里工作了足够长的时间，就让他们在当前团队以外的其他团队工作几周。不论在当前的工作领域还是其他领域，整个职业发展过程中的成长对人们非常重要，所以如果你希望他们留在公司里，就必须为此提供机会。

## 工作负荷

一般来讲，人们都希望工作有一定的挑战性但又能够做到。继续我们关于成长机会的讨论，通过有挑战性的工作，人们可以检验自己的能力，并进一步发展技能，这对于他们的工作满意度非常重要。在第二部分中，我们讨论了多种不同的工作或协作方式，包括开始者和结束者以及纯化论者和实用主义者。如果交给人们的工作常常不是他们最愿意做的工作，这就会让他们感觉工作负荷过大，而对工作本身的不投入也会减弱人们的幸福感，降低生产力。

如果挑战过大也会带来问题。人们可能感觉公司或管理者对他们期望过高，不愿意给他们提供完成工作所需要的时间、支持或资源，或者看起来他们的管理者与现实脱节，根本不了解他们能完成多少工作。这可能说明一个团队承担了过多的工作，或者

工作分配不合理，有些员工相对轻闲，而另外一些员工工作过多。不论是什么原因，如果人们长期工作负担过重（而不是只是短期加班），就会有非常不好的影响。

工作负荷过大的员工最后可能会离开公司，寻找其他不需要这么卖命的工作。即使他们选择留下，但是由于倦怠而身心疲惫，这可能更不好。管理者要定期与整个团队以及单个员工交流，确保他们的工作量不会脱离实际。还要保证员工在需要时可以休假。如果一个团队或个人刚结束加班，如临近项目收尾时，应当鼓励他们放假休息，确保每个人每年至少有一个长假（或者甚至“家中度假”），以避免倦怠。

## 倦怠

倦怠这个词是指长期疲于奔命，对工作以及工作以外的活动都缺乏兴趣。倦怠的症状与临床抑郁症的症状非常相似，在近期的一些研究中甚至就把倦怠称为抑郁症的一种形式。如果人们出现倦怠，可能会把自己与其他人隔离，更少关注自己的个人需要，睡眠出现问题，而且有冷漠、无助和绝望的情绪。这通常是由于太长时间的压力和工作过度造成的，这在技术行业极为常见，特别是硅谷的创业公司，这些公司往往会把“英雄”人物或“明星”开发人员理想化。心理健康与身体健康同样重要，甚至更重要，一定要注意避免倦怠，这应当是每一个团队和每一个公司最首要的事情。

很多技术公司都有某种值班待命的职位，这些人的职责包括接电话或短信，要对正常工作时间以外发生的事件做出响应。要确保待命不会对人们施加不适当的压力，这很重要。如果有可能，要确保至少由两个人分担值班待命的职责，这是最低限度。如果只有一个人没日没夜地待命（24/7/365），很可能会让这个人倦怠。不应该要求任何人无休止地放弃他们的所有夜晚休息时间和周末。理想情况下，应当由多个人轮班待命，随着公司的成长，还可以有多组轮班，使每个人在轮班间隙有足够的时间补觉或放松。

如果有人参与了轮班待命，要确保他们得到相应的报酬。对于负有待命职责的员工，有些公司会提高他们的工资。有些公司里，如果要求员工携带手机，会按小时给予额外报酬，或者对工作时间之外要响应的每一个事件给予额外酬劳，另外有些公司在每次轮班待命后会提供额外的休息时间（几小时或几天）。如果一项工作从一开始就包括待命职责，一定要提前向员工明确这些职责的范围，使员工知道他们要做什么，可以相应地协商他们的报酬。如果这些职责是在员工入职后增加的，要确保员工有机会与管理者讨论相关的细节和报酬。



## 文化和“文化契合”

文化和文化契合这两个词很含糊，可能存在问题，负责招聘的人经常使用这些词，希望（不论是有意还是无意）保持团队某种程度的单一性，他们可能会聘用与他们上过同一所学校、喜欢同一项运动或者参加过同一个兄弟会的人。

这是对文化契合思想的滥用，因为这里只考虑了文化中非常肤浅的想法，并用这些想法营造一种排外的氛围。

由于devops主要关于文化以及人们相互之间如何合作和交互，所以要采用一种有效的方式而不是排外的方式定义这些术语，这很重要。最好将文化定义为人们或社会的理念、习惯和社会行为，深入分析这些领域时，可以看到这个定义会决定人们是否愿意留在一个公司。

理念在一个公司或团队的上下文中可能表示很多不同的东西。最宽泛的理解是，一个公司的理念是它的价值定位，它要销售什么产品，要如何挣钱。有些公司的价值定位实际上是做广告，或者将用户数据出售给广告商。如果有人认为公司的价值观与他们个人的价值观有太大冲突，或者他们对公司所做的事情没有兴趣，不论这个公司是否成功，都会有一种力量驱使这个人离开这个公司。

理念还可以解释为一个组织或团队认为有价值的东西。作为一个典型的例子，很多组织中，尤其是devops思想开始普及之前，运维或IT工作通常很不被看重。IT只被看作是成本中心，总是会尽可能地减少对IT的投入，因为人们认为IT对公司价值不大甚至没有价值。

类似地，有些人可能感觉没有得到团队或管理者的重视。如果有些团队成员与他们的上级非常亲密，而且这种友谊导致上级会更多地听取他们的意见，那么其他团队成员就会感觉自己的工作对团队价值不大。

如果有人与团队中大多数人的想法不同，他们也很容易有这种感觉。我们之前讨论过增加团队或组织多样性的相关问题，不过还要考虑这对于“少数人”可能产生的影响，这也非常重要。如果一个男性团队里有唯一的一名女性，他们就会感觉没有人注意或重视他们的工作，尤其是在主要看谁嗓门最大来处理不一致意见的团队中。

习惯是传统或广泛接受的行为、说话或做事情的方式。从这个意义上讲，工作中的很多方面都可以看作是习惯，包括：

- 如何分配工作，以及谁负责分配工作。



- 一个团队的成员或公司里相同级别的人员相互之间如何沟通。
- 管理者如何向他们的下属传达消息。
- 人们到办公室和离开办公室的时间。
- 完成工作的技术流程。
- 如何升职、加薪和发奖金。

有关习惯的一个问题是，要知道习惯只是做某些事情的一种方法，而不是唯一的方法，但通常很难意识到这一点，因为一旦人们习惯于这样做，这些习惯就会变得熟视无睹。通常需要一个全新的视角才能发现可能有更好的办法来完成这个工作。



认可和重视这些想法很重要，“我们总是这么做”并不是继续这么做的充分理由。如果拒绝做出改变甚至拒绝考虑新想法，会导致团队和公司固步自封，这通常会使他们被其他竞争对手所超越。害怕改变、拒绝不熟悉的事物是人的天性，但是我们应当意识到自己的这种倾向性，并积极努力地努力克服，确保我们能听取、考虑和选择最好的想法，而不是我们最习惯的想法。

如果想提高多样性，就要特别考虑公司关于升职、加薪和发奖金的习惯。需要反复强调一点：这里也很容易滋生无意识偏见，尽管我们可能不希望这样或者可能没有注意到。如果只按管理者的评判来给予奖励，不是由人们申请或鼓励人们申请，或者没有考虑到某个职位或级别的所有员工，就可能出现（而且通常都会出现）无意识偏见。

社会行为是文化的最后一个主要部分，涵盖了人们如何交互的诸多因素。注意人们沟通的方式：更“高级”员工与级别较低的员工谈话时是否显得高人一等或者无视后者？是否所有想法都能同等对待而不论这些想法来自哪里？人们在会议中是否会打断别人讲话，还是等其他人说完后才发言？只是同伴之间是这样，还是与管理层一起时也是这样？人们意见不一致时，他们会如何解决—通过平静的讨论、采用多数人的意见，还是相互之间大声叫嚷，直到某一方受挫放弃？如何做出决策？

社会行为还包括我们听到社交（social）这个词时常常想到的东西。团队如何相互了解或建立维系？更好地了解与你共事的人有很多好处，这包括更有同理心和更有效的沟通，而增进同事友情有一些有效或不那么有效的方法。较大的企业可能选择“破冰”和“信任倒”之类的活动，而创业公司可能更喜欢一起去最近的酒吧。最有效的方法可能是介于两者之间，让所有团队成员发表意见，并努力找出所有人都同意的某种方式。不过，要注意有些人可能不愿意当众发表意见，他们会为此感觉不自在。

例如，有些人正在努力戒酒，可能不愿意在同事面前解释为什么他们不想一起去酒吧参加活动。要确保为人们提供充分的机会，能够以一种安全的方式私下表达他们的意见。

类似地，相对于整体社会行为，人们在办公室的社交方式可能有很大不同。人们通常会注意谁经常与团队经理共进午餐或者一起喝咖啡，特别是并不是团队中每一个人都拥有这样的特权（尤其是创业公司，这些公司通常由之前相互认识的人创立，这些同事可能在工作中成为朋友，更要避免办公室里出现无意识偏见和偏袒，这很重要）。

很多办公室会开展一些活动，尤其是比较小的办公室，这些活动经常在工作之余或工间休息时进行。有些办公室可能有咖啡机或啤酒机，如今这就相当于饮水机。有些可能有桌上足球或乒乓球桌，员工可以在一天工作结束后玩几把释放压力。在主要由年轻男性员工构成的环境中，遥控直升飞机或仿真枪之类的玩具越来越常见。这些东西总的说来没什么坏处，不过对于不喜欢这些玩具的人来说，这会带来一种不好的感觉。有些人可能不喜欢工作环境里有小孩子的玩具，当他正在工作时如果突然被一个瞄错了的橡胶子弹打中头，肯定会大为光火。

要记住，人们不可能不愿意公开反对通常被认为“有趣”的东西，没有人希望被当作是无趣的人。如果某个东西已经成为文化中长期存在的部分，很难有人跳出来反对，而且如果有些人在团队中本来就属于少数派，他们更不愿意出头，否则越发显得他们格格不入。要确保人们能自在地说出他们的想法，还要注意办公室通常开展哪些类型的活动，这对于发展一种让每一个人都有归属感的文化很有好处。

总的来说，要有效地保留员工，很大程度上是要确保与员工维持一个契约，保证大家相互理解，满足共同的目标或需要，并根据前面讨论的devops组合不断培养关系。应当在招聘、面试和保留员工的全过程中尽可能地明确关于文化的期望。牢记这一点，现在来看这些思想在实际中如何应用。

## 案例研究：团队发展和规模化

为了完成这一章的案例研究，我们与两个以不同方式参与招聘工作的人进行了交流：一位是2007年创立的一个在线集市公司的devops主管，另一位是Phaedra Marshall，他是Critical Mass的技术主管，这是1996年在加拿大艾尔伯塔省卡尔加里创立的一家全球数字市场营销和设计公司。尽管他们都密切参与了这些技术公司的招聘决策，不过采用了不同的方法来做出决策。我们将分析他们采用不同方法的原因，并了解为什么这些方法分别适用于他们的特定情况。

## 发展和建立运维团队

这位devops主管最初的职业是开发人员，后来加入一家大型电子商务公司，为在线商店和零售POS系统开发软件，并从头开始建立他们的运维团队。之后他在一家数字媒体和出版公司从零开始建立运维团队。在这家公司里，他从技术方面监督指导生产运维和企业IT部门，并帮助发展这两个部门。

由于他想寻求一个主管职位，另外希望找到一个有充分的学习和成长空间的环境，这家公司以其多样性和文化成功地吸引了他。这家公司一半以上的高级经理都是女性，也包括CTO和CEO，这比大多数硅谷公司（甚至大多数一般意义上的技术公司）都有更大的多样性。目前这家公司有大约125个员工，其中30位在工程领域。

在运维方面，他们在一个物理/云基础设施组合上运行了大约50台服务器，如果负载需要，可以对多达数百个工作节点自动规模化（根据服务器CPU负载等指标自动增加或减少云基础设施中运行的服务器数量）。对他们来说，有效地实现devops意味着让运维工程师与开发人员紧密合作，来构建“漂亮地完成自动化的”系统，帮助企业实现其目标。这包括指导开发人员了解运维工作，尤其是自动化工作，然后在所构建系统的整个生命周期中与他们紧密合作。

### 招聘和面试

为了让招聘决策支持这个devops愿景，这位主管必须在这个新团队的成长策略中加入充分的考虑和迭代。公司的devops团队目前由4个工程师和主管本人组成，每个人的经验有所不同，包括一个初级工程师（这是他的第一个devops职位），还包括一位前主管（退下来成为个人贡献者）。所有这些人的聘用决定都由主管做出。招聘过程包括：首先得到工程VP对招聘人数的确认，然后在推特以及GitHub和StackOverflow求职公告板发布招聘启示。与很多公司一样，通过招募人员来招聘员工的做法不太成功，当前的团队成员发现，特定行业的求职公告板和社交网络对于寻找他们需要的应聘者往往更有效。

面试过程首先进行两轮电话筛查，一次由当前的团队成员初步筛查，另一次由主管本人电话筛查。通过了电话筛查的应聘者会得到一次面对面面试的机会，在这次面试中，他们要与两位工程师、一个工程主管、组织中一位业务经理及最后的工程VP交谈。有些面试比较关注技术，另外一些则更强调对这个人的了解，会对他们是否适合加入这个成长中的团队做出评价。对于这个主管要建立的团队，这意味着要明确他们对于之前的职位喜欢什么以及不喜欢什么，他们想要什么，另外对于工作环境喜欢和不喜欢哪些方面。



这个主管指出，他会努力了解应聘者是否在某些方面有强烈的个人观点（如文本编辑器、选择SQL还是NoSQL，或者他们喜欢的Linux版本），另外这些观点是否过于严苛而没有灵活性。公司发现，有些人拒绝改变想法，这些人往往很难与他们的团队环境相容。让直接团队之外的人（如业务经理）参与面试也很有用，这样可以更好地了解应聘者是否能与其他团队（特别是非工程团队）很好地合作。

## 强观点，弱坚持

Paul Saffo是硅谷的一个技术预报员，他在2008年写的文章中指出，他发现对于涉及中等程度上不确定性的环境（很多技术行业都是如此），处理这些环境的最佳态度是“强观点，弱坚持”。“强观点”是指要确保得出一个结论（如果需要，通常会受其直觉指引），而不是拒绝表态或者根本没有立场。

“弱坚持”是指查看是否有错误，找出与当前结论不契合的方面，愿意而且能够根据新的证据改变想法。很多人发现这是应聘者应有的一种态度，因为没有强观点的人不会成为强有力的决策者或领导人，但是如果过份坚持自己的观点，这些人可能会使自己陷入困境，而且不太容易从错误中恢复。

## “英雄文化”的问题

有人指出他们的招聘启示在宣扬英雄文化，其中把“英雄”行为看作是必要的，如长时间工作、独立地排查问题，以及不断“救火”来保证服务持续运行，这位主管意识到他们的招聘过程需要做一些改进。

这种文化的问题在于它是不健康的，长时间的工作和周末连续工作会导致倦怠，不仅身体吃不消，通常还会导致精神健康问题。另外，这会吸引一些希望成为“英雄”的人，他们会对自己的成就或收获更感兴趣，而不关心是否能作为团队的一部分有效地工作。尽管招聘启示中对英雄文化的宣扬是无意的，但是会影响这些职位吸引到什么类型的应聘者。

这种招聘启示的例子包括：

- 要求应聘者有“110%”或“超水平”的表现，这些说法往往暗示着这个团队很少甚至根本没有工作-生活的平衡。不仅不健康，这种要求还会偏向单身的人，而拒绝有家庭责任的应聘者（或者有些人只是希望大多数晚上能回到家里）。

- 描述一个团队“努力工作，努力玩乐”。你聘用的是员工而不是朋友，希望你的员工把他们的非工作时间都用在在工作社交活动上，特别是如果这些活动需要大量喝酒，这会让很多应聘者无法接受。
- “非常棒”或另外某种含糊的特点。类似这样的词很含糊，几乎毫无意义，只会吸引那些认为自己“非常棒”的人，这通常与自大和不愿意学习或倾听关联在一起，而把饱受“无法相信个体成功是自身实力的结果”之苦的人群排除在外。这包括要求应聘者描述自己的技能时把自己描述为“明星”、“高手”、“奇才”等。
- 布置作业或其他工作，要求应聘者证明他们的知识水平，这也是一种不好的策略，会表现出对员工的时间缺乏尊重，这往往会偏向那些在工作之外没有太多其他责任的人。并不是所有筛查练习都不好。例如，如果要求预演假想的场景来了解一个应聘者的思维过程和价值观，这可能很有好处，但是如果过于依赖问题解答，可能会让应聘者不能接受。

诸如此类强调英雄文化的要求往往会带来这样一种工作环境：员工总是废寝忘食，这本身会造成创造力、生产力以及同理心的下降，最终会导致对工作不满意，丧失自信和倦怠。

## 招聘启示和招募问题

仔细检查他们的招聘启示时，公司详细查看了招募策略整体是如何描述的。他们维持了原先不使用招募人员的决定，因为大多数情况下，招募人员（特别是第三方招募人员）可能还会带来问题，如果招募人员与你的价值观不同，就不能很好地代表你的公司，可能会让应聘者不愉快甚至被冒犯。下面的例子不是来自这个主管（也不是出自这家公司），不过说明了招聘启示或（内部或外部）招募人员消息中特别要注意的一些问题。

### 不注意细节

如果一个email开头是“亲爱的%%FIRSTNAME%%，我们需要一个胜任%%JOBTITLE%% 职位的人”，可以清楚地看出，这是从一个模板复制粘贴来的，甚至在单击发送前可能都没有检查过这个email是否有明显的错误。即使从LinkedIn个人简介中复制一个潜在候选人的技能也不保险，特别是如果没有做过检查校对，就很可能有错误；如果给某个人发出一个email要求讨论他们“关于后台开发和喝啤酒的经验”，这说明这个人没有仔细检查，或者在为一个人很不健康的公司文化招聘人员。如

果向团队中的每一个成员复制粘贴相同的格式信件，这会表现出公司的懈怠，团队成员不会注意不到，而且采用这种策略的公司会有很不好的名声。

同样地，如果一个潜在候选人对这个职位不感兴趣，就请他们推荐其他人，这也是很懒惰的做法。如今，几乎每个公司都在积极的招聘人员。如果他们知道一些合适的候选人，会亲自与他们交谈，而不是为其他人的公司做免费招募工作。人们通常是愿意提供推荐的，但是只针对朋友或他们了解和信任的同事，而对于反复通过群发email骚扰他们的陌生人，人们不会有兴趣为他们提供推荐。

## 排外或非专业语言

要注意招聘启示或email中可能让应聘者反感的语言。有些男性化语言（比如“代码”、“明星”和“你是个‘技术’男吗？”）可能会让一些人感觉不舒服。公然的性别偏见的言论则更糟糕。如果招聘启示里写着一些带偏见字眼的话语，从专业角度讲这是很不合适的，这会暗示这个公司很可能对某些人不友好。假设你要找多少岁以下的工程师，这也会有同样的效果，而且有些地方限定性别和年龄通常是不合规的。

## 对技术的过度关注

很多工程师非常热衷于使用新技术，所以公司可以利用这些技术来吸引人们关注他们给出的职位，这是有道理的。不过，如果过度强调技术，尤其是没有开展适当的调查，可能就会出问题。如果要求应聘者对某种特定技术有一定经验，你首先要做一些研究。如果要求对某个产品有10年的经验，但是这个产品问世才只有2年，看起来你完全不知道自己在说什么，你肯定不希望给人留下这样的印象。

还要注意的是，应聘者通常更关心一个公司做什么，而不是它使用的技术。如果你的技术确实很有意思，那么确实可以提到这个技术，但是如果向一个可能的应聘者发email只谈技术，而对于要用这个技术构建什么以及这个公司做什么甚至只字不提，这绝对是错误的。一个工程团队总是使用最新最“热辣”的前沿工具并不一定能很好地推销这个团队（同样地，诸如“热辣”和“性感”之类的词很可能让大批潜在的应聘者敬而远之）。



### 检查你的招聘启示

检查是计算机编程中的一个术语，描述通过编程搜索可能会导致问题的可疑、危险或不可移植的代码构造。工程师可以“检查”他们的代码完成这种分析，查找常见的错误或样式问题，然后才把代码提交到主代码仓库。

可以用一个类似的工具分析招聘启示或者招聘邮件，检查我们前面描述的一些



常见问题。你可以自己使joblint.org上有这个工具，捕捉你没有注意到的一些问题，还能提醒你将来应当检查什么。

这个公司随后修改了他们的招聘启示，去除了这些“英雄主义”的描述。现在他们强调团队的文化价值观，包括工作—生活平衡。例如，他们提到，对于每一个值班待命的工程师，在一周轮班之后，会额外给一天假来帮助缓解因值班带来的压力和睡眠不足。他们还努力优化运维工作中不合适的部分，采用类似人员轮流换班的措施，即人们轮换着负责回应突发问题和同事临时提出的其他问题，并使用一个ticket支持系统跟踪这些请求。

下面是一些更好的招聘启示的例子：

- 提到一般技能而不是特定技术。不要说你希望一个人有2年的Puppet经验，而应当要求应聘者了解类似重复性任务和配置管理自动化等概念。另外要评估是否需要指定年数的经验。很多情况下，这并不需要，而这些严苛的需求会把本来合格的应聘者排除在外。
- 强调重要的文化价值观。谈到文化，我们并不是在说要有一个一起喝酒一起踢足球的团队，而是指诸如同理心、有效的沟通、去除孤岛以及工作-生活平衡等文化价值观（当然，不要提你的团队所没有的价值观。如果对你的文化有所隐瞒，聘用的人很快就会发现，而不好的名声也很快会传开）。
- 确保招聘启示是性别中立的，没有冒犯性的字眼。你要找写代码的人，那就直接讲“写代码”，而不要用一些粗俗的说法。
- 特别强调公司对多样性的重视（如果这是你努力的一个方面）。要提到可能对更大范围应聘者有吸引力的福利，而不只是有一个放啤酒的冰箱和乒乓球桌，所崇尚的文化应当鼓励人们按时下班、有产假并提供培训机会。



### 其他多样化招聘资源

《Model View Culture》是一个有关文化和多样性问题的独立出版物，列出了关于多样化招聘的25个技巧。这组资源非常好，如果想要提高团队的多样性，很有必要了解这些资源。

这个新招聘启示的效果非常好，这位主管为他的团队成功地招聘了5个人，其中只有一个人最后不太胜任。正是依靠聘用的这些人，他们努力将基础设施从不可管理的“雪花”服务器改为完全自动化的服务器。他们的团队大力聘用对自动化和测试充满

热情的工程师，然后让他们尽其所能的工作，而不是事无巨细地进行管理。这使他们有了全新的自动化和测试基础设施，可以为整个工程组织提供使用简单、文档齐全的工具，每个人都知道如何使用这些工具以及如何为之做出贡献。

## 个人和团队发展

现在我们转向Critical Mass，这个公司的技术主管Phaedra Marshall在不同行业的技术领域有15年的工作经验，包括高等教育、财务、媒体和广告。对她而言，devops意味着充分利用开发人员编写代码的技能，以及系统管理员的运维知识，从而大规模地供应和运维可靠的计算系统，不论这些系统支持的特定行业是什么，这都非常重要。她希望基于现有的招聘启示和职位创建一个环境，能够让这两类技术人员充分发挥他们的强项。

作为团队领导人，她的关注点是发展和提高她的团队，这与之前介绍的主管类似，但是由于Critical Mass的员工人数超过700人，它的规模远远大于之前的在线集市公司。这就要求采用不同的招聘方法，尽管总目标是一样的。

由于他们的组织如此庞大，所以配备有专门的招募人员，这些招募人员属于公司内部人员，他们的工作是全职的，就是努力寻找合格的应聘者。这使得这个团队可以与这些招募人员紧密地合作，确保他们与团队和公司有一致的愿景，而避免之前提到了招募人员可能带来的一些问题。如果一个团队领导人或招聘经理对从招募人员那里得到的候选人不满意，他们要与指定的招募人员合作来努力解决那些问题。

面试过程包括一个电话面试，这通常持续大约30分钟，主要介绍应聘者的工作历史，后面会有一些现场面试。具体的面试次数取决于应聘者所面试的团队和职位类型。例如，一个入门级开发人员可能只要求一次现场面试，而应聘者申请的职位越高，他们要经过的面试次数就越多。

## 团队成员发展和成长

一旦完成了面试过程，决定聘用，会为每个员工指派一个职业开发师。对员工来说，所指派的职业开发师就相当于导师，通常一个职业开发师指导4个员工。一段时间后，公司发现如果职业开发师指导的人数超过7人，就不能很有效地工作，因为他们还有其他的工作安排和职责。职业开发师的主要目标是帮助员工在公司里取得成功，不论采用何种形式，因为他们宁愿一个员工在公司里改变职位或团队，也不希望他由于对工作不满意而离职。每个职业开发人员每个月至少要与所指导的员工单独会见一次。

这个公司采用一种“360反馈机制”完成绩效审查。采用这种方法，会在一个员工的工作圈子里从各种不同的人那里收集反馈，通常包括他们的直接伙伴、他们的直接下属（如果有）以及（一个或多个）上级。在这个案例研究中，360反馈是匿名提供的，职业开发师会收到指导对象所有反馈的一个副本。采用这种方式，职业开发师可以帮助指导对象处理这些反馈，例如，如果有严重的负面反馈，可以提出一个绩效改进计划，或者帮助员工对他们的短期和长期职业目标做出具体的计划。这是职业开发师帮助指导对象的一个重要部分，另外还会提供一般的职业方向建议或者帮助他们解决他们困惑的技术问题。

Critical Mass非常关注职业发展和指导，由此可以看出，这家公司将保留员工看作是其招聘和聘用策略中很关键的一部分。他们意识到，尽管货币报酬很重要，但是除了加薪以外，还有其他方式可以让员工感到自己的重要性。

在他们的每周技术人员会议中，会鼓励所有团队成员谈论他们正在做的工作，并介绍他们对这些项目做出的贡献。每个月都会让员工提名一个同事得到突出贡献奖，一方面可以让人们感受到来自同伴的尊重，获得人际关系满足感，另一方面还能通过奖金获得认可。

要允许员工有多种兴趣，并允许在他们选择的方向上取得职业发展，这也是保留员工的一个很重要的策略。技术领导人分享了她团队中一位前端开发人员的故事，在这个团队中，这个开发人员很受重视，不过他希望除了JavaScript和CSS还能做些其他的工作。他与他的职业开发师讨论了这个问题，职业开发师让技术领导人注意到这一点，然后他们三个人共同提出了一个办法：让这个开发人员每周至少花25%的时间参与其他创造性技术项目。尽管只是对他的职责做了小小的改变，但是这会帮助他发展技能，使他有机会在感兴趣的技术领域工作，而且公司也留住了一个有才能（而且更快乐）的开发人员。

从这两个案例研究可以看到，尽管两个公司有相同的总体目标（招聘和保留有才能的工程师），但由于公司规模和特定情况不同，他们采用了不同的招聘和保留方法。不论规模大小，这两个组织都强调了他们希望创建和保持的文化，并建立相应的招聘和保留策略来满足这些目标。



如果只聘用对某个特定技术有经验的人，这就类似于只看本书中关于工具的那一部分而跳过所有其他内容。要有效地实现devops，这些强调技术的“硬技能”只是其中很小的一部分。假设聘用的一个人之前很好地补充了你的技术堆栈，但是在文化契合方面很差，或者缺乏批判性思维、学习和问题排查能力，



如果你将来需要改变技术，或者要在你的技术堆栈中增加一个新技术，会怎么样呢？

另一方面，如果你发现一个开发人员有卓越的人际交往能力和学习能力，从长远来看，他学习新东西的能力将是一个更突出的优点。在壮大和发展你的团队时要记住这一点，这会使你的招聘方法和整个组织都更为有效。

## 团队规模化和成长策略

在某些方面，可以把团队看作是一个单独的有机体。在一个规模变化的组织中，要确保建立健康的团队，协作和亲密性方面的技能至关重要，你可能还想再看看本书的第二部分和第三部分。另外还有3个关键的策略，可以利用这些策略在团队的整个生命周期中增强团队：

- 保持团队小而灵活。
- 发展协作。
- 管理冲突。

接下来我们来看这些不同的策略如何影响团队健康和生产力。

### 保持团队小而灵活

随着组织的成长壮大，团队成长时可能没有周全的设计。大团队不太容易分享知识或者相互自由学习。由于不熟悉团队中各个成员的优缺点，这会导致不能灵活地分配工作。任务会对应到个人，这通常会带来瓶颈，尤其是在复杂情况下人们之间可能形成循环依赖。

目前你的环境状况如何？在特定的主题或服务方面是否存在单一知识点？这是小型创业公司中经常出现的情况，为了完成任务，人们会承担多个角色和职责。随着团队的成长，需要监控这些单一知识点。这是你的环境中隐含的脆弱性，如果这个人决定离开你的公司，或者更糟糕的，由于某种重大事件导致他们不能继续留任，可能导致额外的人员流失。

20世纪60年代，心理学家Frederick Herzberg采访调查了多位工程师和会计人员，提出了关于工作满意因素和不满因素的一个激励保健理论（见图14-1）<sup>注5</sup>。

---

注5：Christina Stello, Herzberg's Two-Factor Theory of Job Satisfaction: An Integrative Literature Review (Minneapolis: University of Minnesota, 2011)。

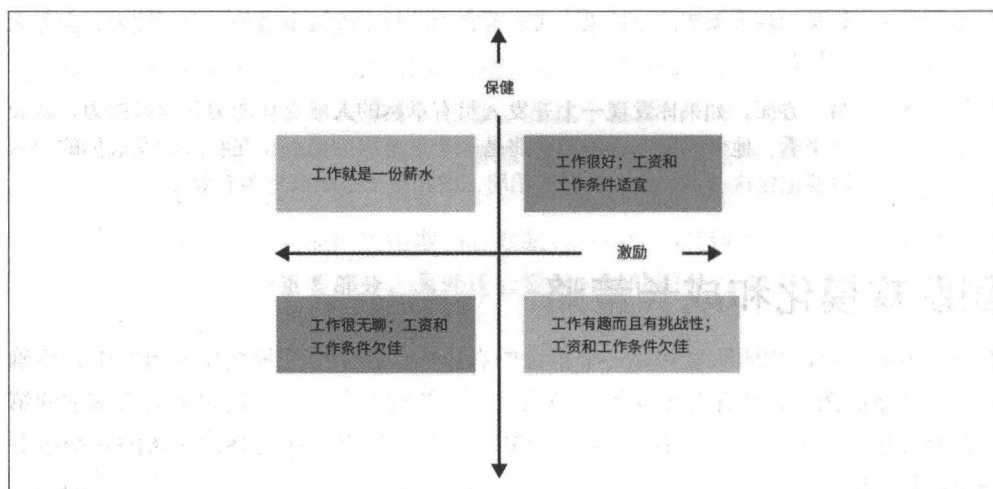


图14-1：激励保健理论

激励因素可以提高工作满意度，包括挑战性的工作、对成就的赏识和认可、感知到的责任和自治性，以及有机会完成有意义的工作等因素。有5个关键要素会对人们产生推动和激励：

- 自由。
- 挑战。
- 教育。
- 有意义的贡献。
- 积极的环境。

保健因素可以消除对工作的不满意，包括工作安全性、公平的工资、福利、工作条件和假期等因素。

在Herzberg的激励保健理论基础上，J. Richard Hackman和Neil Vidmar在20世纪70年代对最佳的团队规模进行了研究。<sup>注6</sup>他们成立了从2人到7人不同规模的团队，来评估在多个不同任务中团队规模对流程和绩效的影响。任务完成之后，他们向参加者询问了对团队规模和所分配任务的感受。根据这项研究，他们发现最佳的团队规模为4.6

注6： J. Richard Hackman and Neil Vidmar, “Effects of Size and Task Type on Group Performance and Member Reactions,” *Sociometry* 33, no. 1 (March 1970)。

个人，所以要限制你的团队大小，使团队人数接近这个数。有些团队要做大量服从多数人意见的决策，根据这一点，可以选择一个奇数，如3或5。

Hackman和Vidmar还指出团队成长到人数达到两位数时影响正常工作的一组因素：

- 沟通和协调事务的成本呈指数增长，导致花在工作上的时间更少。
- 更多的转手导致更多错误和误解。
- 团队整体凝聚力减弱。



经过一段时间后，一个团队能更有效地完成当前的工作，因此可能会增加更多职责或者改变工作方向。团队达到某种规模时，如果开始影响有效性，而且要求另外增加人手来完成工作，就需要考虑拆分团队，或者减少团队的职责。

团队应当根据需要进行调整，来维持团队内部以及团队之间的有效性。规模小的组织允许交互式的团队沟通，因为团队中的每个人都相互认识，所以沟通是高带宽的。随着组织的成长，每个人的带宽会降低，直到最终组织中的每一个人都相互不认识。应当保持小的团队规模，以维持足够的沟通和共同的理解，从而支持组织中的devops组合。

## 发展协作

在第二部分中，我们介绍了理解个人背景、目标、认知方式和思维模式的重要性。我们还讨论了影响团队的组织压力，以及解决冲突采用的协商策略。在这里我们将重点讨论面对规模化挑战时决定团队成败的关键因素。

来自领导和管理层的支持对于保证有效协作至关重要。不应该有影响人们积极性的绩效考核项目，如分级评级，不过，更好的做法是好的行为应当得到激励。

研究优秀团队时，David Engel和助手们发现，最有效、最有生产力的团队往往由“经常沟通、平等参与而且情绪解读能力强的”成员组成。<sup>注7</sup>研究远程团队时，本地团队中观察到的群体智慧在远程团队中至关重要。不论是本地还是远程团队，拥有这些重要技能的团队总是在有效性和绩效方面更高一筹。我们将这些发现分解为高效协作团队的以下特点：

注7： David Engel et al., “Reading the Mind in the Eyes or Reading Between the Lines? Theory of Mind Predicts Collective Intelligence Equally Well Online and Face-To-Face, PLoS ONE 9, no. 12 (2014).



- 团队成员积极的相互依赖。
- 有效的沟通。
- 个人和团队当责。

## 团队成员积极的相互依赖

团队中的相互依赖体现为相互之间的一种信任和责任。在一个新团队中，需要一定的时间才能建立这种信任和尊重，每一个新的团队成员加入或有成员离开这个团队时，都必须重新建立信任。

一段时间后，人们会了解团队中其他成员的优缺点，并了解如何根据上下文环境和团队成员是否有空余时间来安排活动。有效的头脑风暴允许团队平等地参与，发现新想法，然后研究这些想法并确定要采用哪些想法。

创建快速团队是从参加编程马拉松衍生出的一个技能，与工作环境中的传统团队相比，这个团队要在相当短的时间内建立、合作然后解散。会有一个明确的最后期限，而且大家相互信赖，共同完成项目。成功的团队会快速找出团队的优点和缺点，寻找能够平衡团队的其他人员。

研究表明，在工作环境中，个人可能有3种行为方式，作为给予者、接受者或匹配者：<sup>注8</sup>

- 给予者，给予多于接受的人，帮助其他人而不期望回报。
- 接受者，接受多于给予的人，只是在对自己利大于弊的情况下才有策略地施以援手。
- 匹配者会在这两者之间，努力维持给予和接受的平衡。

团队中的每个人都有一组互惠偏好，但这也可能导致团队中的个体发生冲突。



## 了解何时和如何寻求帮助

在一个团队上下文中，要考量什么时候上下文不明确，并以某些方式做出响应来得到额外的信息，这一点至关重要。个人基于单独的上下文建立理解并完成工作是最容易的。不过更长远来看，这是有成本的。如果人们所在的公司是从创业公司逐步发展成为成熟的企业，对他们来说，最困难的事情之一是这会随

注8： Adam M. Grant, Give And Take (New York: Viking, 2013)。

上下文改变，而且需要面向团队和组织的目标，而不能只根据他们孤立的理解来完成工作。

在一个群体中，如果人们还没有形成强有力的团队纽带，询问更多信息或请求帮助可能会被误解而且不受欢迎。资深的员工可能更难寻求帮助。可能有这样一种误区，认为随着我们职业生涯的发展，我们可以而且应该在所有方面都是专家。

人们不愿意寻求帮助的另一个原因是不希望带来义务。寻求帮助的行为可能会对某个人存在义务或相应的成本。在一个健康的组织中，人们不会“在意”这种义务，而会关注为了对团队和企业整体有利需要做什么。

主要有3类给予（见图14-2）：

- 时间。
- 知识。
- 金钱。

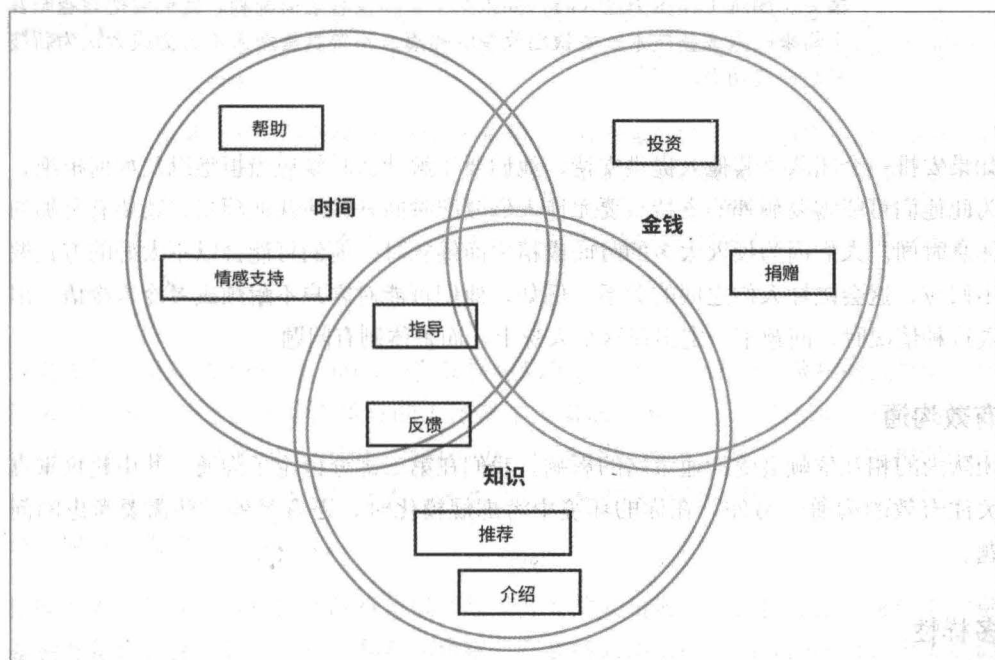


图14-2：给予行为类型

我们的大多数活动可能都在这个范畴内。查看对个人的总体价值和成本时，按这3个维度来考虑会有帮助。例如，指导和给出反馈的活动可能比提供介绍的成本更高。

考虑如何做一个更好的“给予者”时，可以考虑你喜欢的领域，以及你在哪些方面可以从给予中受益，并重点关注这些方面。有时接受者并不需要你为他们提供的帮助，所以不要在这个方面浪费你的资源。例如，如果有人需要得到情感支持，这不表示他们需要或想要你的反馈或个人经验。



## 挑战和改变

如果不问问题，会减少保持相互理解和共同点的机会，如果对企业中的低级人员设置一个行为模式，这种情况会更加恶化，更缺乏理解，他们必须浪费宝贵的学习和行动时间重做重复的工作。

寻求帮助的第一部分是重述你的假设，以明确大家的理解。在一个已经形成强有力纽带的环境中（可以回顾我们在第一部分介绍的攀岩例子），“belay形式”的交流尽管简洁，但足以让两个人理解当前状态。

新的项目或团队成员与新的管理层需要额外的沟通，直到投入了足够的时间和精力。团队工作压力过大时，如果某个工作没有之前顺利，他们会把问题归咎于新来的人或新技术。要找出改变的痛点。不要责备新人不努力或者认为新技术是白费功夫。

如果安排一个团队为其他人提供支持，他们要了解什么时候应当拒绝以及如何拒绝，为此他们可能需要额外的支持。要允许人们分配时间来关注其他项目，还要有足够的休息时间。人们因为投入太多的时间或精力而倦怠时，他们可能会以不太好的方式做出回应，这会破坏人们之间的关系。例如，他们可能对客户不耐烦或者冷言冷语。出现这种情况时，问题不一定出在这个人身上，而是体制有问题。

## 有效沟通

团队内的相互依赖会受沟通策略的影响。我们在第二部分讨论了沟通，其中建议重点关注有效的沟通。另外，在你的环境中考虑规模化时，还有另外一些需要考虑的问题。

## 多样性

如果幸运（当然要有必要的重视和努力），你的团队会在成长过程中越来越多样化。如第二部分和第三部分中所讨论的，从长远来看，增加多样性会增强同理心、创造力



和问题解决能力，但是短期内可能导致人际冲突的增加。由组织拐点产生的额外压力肯定会增加这种紧张氛围。

有一点很重要，要确保团队经理了解这种难度，而且在包容性和无意识偏见方面已经进行了培训。随着组织的成长，团队或部门有专门的HR代表可能会很有帮助，这样个人贡献者和他们的管理者出现问题时就有了指定的联系人。

## 个人和团队当责

个人当责是指能够根据一个人自己的动作和行为确认结果并对结果负责。如果人们知道同事完成了高质量的工作，这会增加团队凝聚力和认可度。如果被问到“你的工作是高质量的吗？”大多数人都会回答“是的”。一般来讲，如果给予人们工作的自由，他们会尽其所能做到最好。如果团队中所有人都可以说“是的，我的同事完成了高质量的工作”，这意味着团队中存在信任、共同的价值观和团队凝聚力。



### 定义质量

在团队和组织中明确地定义质量很重要。要让客户了解你如何定义质量，这个过程要透明，并鼓励他们分享对质量的理解，这会使产品结果与目标更加一致，确保产品的可操作性和性能，并促进学习。提供明确的质量指标也有利于个人当责。

另外，对于一个组织中的各个角色，工作质量的定义可能有所不同。例如，对于写代码的团队，由于重点是高质量的代码，这会使他们在一个版本上花更多的时间，以至于妨碍同样需要时间的其他团队。要得到关于质量的共同认识，明确对于不同的角色什么更重要，这样在解决问题时有助于减少不必要的冲突。

质量圈最早由W. Edwards Deming在20世纪50年代提出，表示一个做相同或类似工作的人群，他们对于什么是高质量的工作基本上看法一致。让人们在这个方面达成共识很重要，这可以增加一个团队的内聚力（因为团队成员经常完成相同或相似的工作），另外由于对高质量的工作有一致的定义，朝着这个共同的目标努力还可以尽量减少误解，支持devops组合。

团队当责是指能够根据团队的行为确认结果并对结果负责。个人接受团队责任的关键是相互信任，并对相互的行为和动作负责。在高绩效团队中，出现问题时，个人会以尊重的态度面对并彼此支持。查找问题与讨论问题之间拖延的时间越短，就能越快地解决问题。问题存在时间越长，不信任就会滋长，人们可能越发不愿意作为一个团队协同工作。

## 管理冲突

在第二部分中，我们讨论了协商策略和冲突。在公司的发展历程中，会形成很多根深蒂固的习惯和行为。为了保证团队的健康，管理层必须帮助人们将自己的个人愿景与公司 and 团队的愿景一致起来，鼓励个人和团队发展技能，并对符合规范和期望的行为提供激励。

在某些方面，冲突可能对一个团队是有益的。这会向团队展示新的想法和观点。如果一个团队里没有任何冲突，这说明存在单一性，需要加以解决。为组织带来价值是我们的义务。如果我们是可有可无的，那么被组织剔除时就不要感觉奇怪。妥善解决冲突的技能对于个人、团队和组织的成长都是必要的。



### 对期望行为做出表率

在团队会议中对你希望看到的行为做出表率。当人们看到管理层只是在抱怨同伴的问题，而不是直接解决这些问题时，他们会复制同样的行为模式。这一点也适用于所有担负指导或领导岗位的个人贡献者，尽管你不是管理者，但是由于你作为指导或领导的地位，人们很可能在工作行为方面寻求你的指导并采用你的方式。

另一方面，冲突并不表示容忍或鼓励霸凌行为。霸凌行为要从环境中完全消除，不能接受霸凌可能带来的价值，这是不正当的。这些行为往往针对某些目标，可以从这些目标的感受来识别霸凌行为，这可能包括压抑、羞辱、消极或自卑。如果有霸凌行为的人本身在一个领导或管理岗位上，这会更糟糕。以下是霸凌行为的一些例子：

- 责备个人犯错误。
- 批评某个人的能力。
- 威胁会失去工作。
- 使用无礼或贬损的方式。
- 贬低或不承认成就。
- 采取排外的策略。
- 大喊大叫。

这一类行为说明存在一个病态或不健康的工作环境。如果一些技术很强的人有这些行为，很可能被忽视，尤其是还处于成长阶段的较小的环境中，往往会认为一个人的硬技能或技术技能更重要，即使他们缺少软技能或人际交往能力（或者甚至有侮辱性的行为）也无关大碍。不过，由于这些个人可能带来的影响以及这种影响会在更大的组织中传播，允许或鼓励这种霸凌行为可能会带来一个越来越“有毒”的环境，一段时间后，这会越发不健康并降低生产力。

## 团队内部的冲突

在团队中，可能有很多不同的冲突来源。在这里我们将讨论几种最常见的冲突，并介绍可以在个人或团队层次上应用的一些解决方案。

### 与团队目标一致

前面讨论过，不是每个职位都适合每一个人。人们有不同的个人喜好、动机和工作方式，这很正常，因为多样性正是团队和组织保持弹性的一个重要部分。不过，这也使人们发现他们所在团队的优先工作或工作方式与他们自己的方式不一致（尤其是在较大的组织中）。

人们可能发现他们与团队中的其他人格格不入，这可能是由于对团队的优先工作或项目缺乏兴趣，或者与同伴或管理者有个人冲突，也可能由于工作方式不一致。通常每个人都希望这个问题得到解决，因为由于这种不一致导致的负面态度、冲突增加或者工作质量降低最后会对团队中的其他人产生负面影响。可以与同伴和管理者经常性地定期面对面交流，这有助于尽早找出和解决这种问题。



在我们的经验中，经常会遇到力求完美和主张执行的不同想法，这在第二部分的讨论中称为开始者和结束者。在这个方面，可以转向不同的项目，或者转到同一个项目中的不同任务，这对于尽可能减少冲突很有帮助。

大组织的好处之一是通常有更多的机会可以让人们在不同的团队甚至内部部门之间转换。在第三部分中讨论过，如果为人们提供机会，允许他们参加训练营或者在不同团队之间轮岗，这不仅能帮助他们找出不一致，还可以发现他们更适合的团队，应当鼓励而不是打击这种改变。

### 与组织目标一致

类似于与团队的不一致，人们可能发现自己与组织整体及其目标并不一致。这往往出



现在某个拐点之后，如一次合并、收购或裁员，另外在组织生命周期中的任何其他时间点也有可能发生。例如，有些人在比较小的创业公司工作很得心应手，一旦他们的组织成长到超过某个规模时，他们就会越来越不适应。

个人与组织整体的价值观如果不一致，这比与团队的不一致更难克服。有一点很重要，要认识到如果出现这种问题，这不一定是由于哪一方有错误或者做得不对，要以不伤害感情的方式帮助人们转到一个更合适的地方。这样一来，将来他们更有可能为你的组织说好话，向别人推荐你的组织，而不会过河拆桥。

## 激励不一致

除了目标、优先工作或工作方式的不一致，激励也可能随着组织、团队和个人的成长和改变而变得不一致。同样的，这更有可能在某个拐点事件之后发生，如合并或收购。

同样的，定期的一对一面谈会很有用，可以帮助发现这种冲突，如果当前以钱作为激励时，HR代表会有很大帮助。不过，这里可能会遇到一些问题，如有些人以钱作为激励而他们所在的组织正处于衰落阶段，或者他们以高质量的工作作为激励，但是团队或部门的其他人不是这样。可以在团队或组织层次上处理激励问题，不过无论采用何种方式，关键是要尽可能快地解决，否则负面情绪和消极情绪会从个人传播到团队的其他人。

## 团队之外的冲突

团队内部肯定会发生冲突，不仅如此，随着部门或组织规模的扩大，跨团队的冲突越来越普遍。实际上，团队之间的冲突正是推动devops运动的主要因素之一。这种冲突常常源于不同团队动机或期望的不一致。

## 不切实际的期望

如果有不切实际的期望，团队的管理层要对此负责。另外要由管理层确定一个给定团队的期望是否实际上是不现实的，尤其是那些有较多新人或初级成员的团队。

一段时间后，团队可能发现对他们的期望变得更不现实。例如，如果你在使用一个指定的运维模式（如第三部分和第四部分中讨论的模式），随着组织的发展，需要支持更多的团队，对你的运维团队会有越来越多的期望。如果只是期望在增加，而运维团队的人数并没有随之增加，这些期望或需求很快就会变得不现实，这就要由团队经理来处理。个人贡献者在组织中的影响力不如他们的管理者，所以不具有这样的能力，需要考虑到这一点。

## 评价团队一致性和能力

如果对一个团队的期望很现实，但是这个团队经常无法满足这些期望，这可能有很多原因。最直接的结果是，这个团队最后会失去其他团队对他们的信任。这个团队是否会承担责任？我们在这一章前面讨论过团队和个人当责，不过要记住，更大范围的组织改变可能导致个人以及团队的不一致，必须查找、明确并解决这些不一致。

团队的工作是否超出其能力范围？要确保团队在做最重要的工作，而且是对公司有价值的工作。另外，注意对这个团队的期望是否现实，尤其是如果近期组织有很大发展，更要注意是否会过度增加对团队的期望。如前所述，devops不是要让相同数量的人完成两倍的工作量，所以要确保团队配备足够的人力来满足对他们的期望。时间、精力和人员要求都必须满足，团队才能满足它自己的需要。

## 组织规模化

组织的规模化会有更大的挑战。需要在很具体的层次上做出决策，适当的团队以及个人要掌握数据。这要求充分的协调以及数据透明度，从而有足够的数​​据支持这些决策。单独工作的独立团队会构建一些工具，这些工具可以帮助其他人，但主要是满足这个团队自己的需要。

## 集中与特定团队

集中团队来提供支持功能会导致倦怠，会使某一个团队成为所有其他团队的全部依靠。高效的支持团队会让工作显得很平常。如果支持团队的价值没有体现并在组织中为人所知，人们就会认为支持团队没有多少价值，尤其是在存在声望阶梯的组织中。这会大大伤害士气，而且一段时间后，这个团队的效率会下降，而这会进一步影响整个公司。

特定（Ad hoc）团队鼓励个人协作，从跨职能的工作到设计、构建以及沟通决策都可以协作，这就允许有多种观点，而且很容易调整。这还允许人们跨越种族界限。



我们无法度量一个人的能力，无法衡量一个人对一个组织有多大意义，是多么不可缺少。如果我们没有充分了解他们的价值，一旦他们离开，所带来的影响会清楚地显现出来。要注意有些决策可能会建立单一责任点，这会导致组织内部的脆弱性。

## 建立领导

要建立一个协作的领导团队，从而能推动日常改变，应对出现的机会和挑战，并监控关键路径。要由整个领导团队共同完成这些不同的任务。可以利用一些支持任务指标收集的工具，从定量和定性方面更好地确定这个团队需要的人。

不论这个组织正在经历规模化的哪个阶段，应对出现的挑战时，领导层需要展示一些关键的行为。这些表率行为包括当责、关注和跟进。如果领导层有一致的行动和不断增强的价值观，这会帮助建立一个健康的组织。

### 当责文化

一般地，当责是对责任的确认和假定，在组织层次上以及团队和个人层次上都是如此。我们可以从一个团队或个人的角度考虑当责，这通常包括对项目 and 单个结果负责，以及对学习和开发活动负责。最后，我们还可以从领导角度考虑当责，这往往会有额外的财务和制度责任。

不论从什么角度考虑，都要问一个重要的问题“谁决定我们要当责，以及谁决定我们要做什么？”通过回答这个问题，你就能明确谁定义当责以及为什么当责很重要。当责文化包括清晰合理的期望、高绩效的正面结果以及低绩效的负面结果。

在第三部分中已经提到，关于当责的一个重要的误区是认为这是要追究责任进行惩罚，或者认为鼓励当责会导致一种畏惧文化。忽视绩效问题和避免当责会鼓励人们采取规避行为，失去相互之间以及对管理层的信任。组织会培养畏惧文化，影响个人的当责能力。如果人们因为某个问题被责备，而没有必要的权威来予以修正或改善，就会从当责文化转变成畏惧和责备文化。

关于当责的另一个误区是个人很自然地要对自己负责。即使在自我管理或自我修正的团队中，领导也需要树立和鼓励好的行为。过份强调个人当责而没有澄清企业目标，可能导致错误和误解增加。由于组织的复杂性，会有大量彼此竞争的目标和需求，这使得自我修正和自我当责更为困难。

这并不表示人们不能和不会自我激励，但是如果管理者期望他们的下属主动地找他们解决问题或错误，这就会留下责任缺口。在从一个问责组织转变到无问责组织时尤其是这样：如果人们已经习惯因为错误或低质量而受到惩罚，他们很自然地会害怕提出这种问题。人们需要得到指导和引导，了解如何对自己以及其他负责人负责，而不会退回到问责或畏惧行为。



## 组织灵活性

一般认为更大的组织（尤其是已经发展多年的组织）改变和适应会慢得多。很多情况下确实是这样，如果考虑受影响的人数，与只让不到一百人改变相比，要让成千上万或者上百万的人完成同样的改变，肯定要花更多的时间和精力。

作为一种更敏捷的软件开发风格，好处之一是可以更快地完成更改，而且由于有更短的反馈周期，这也意味着可以更快地更改，而不是之后响应新信息时才做出更改，这可以减少浪费时间和精力。一个组织的灵活性可能很大程度上取决于另一个因素：团队如何组织以及影响其交互的过程。

考虑一个大型组织的灵活性时，常问的问题包括：

### 团队之间人们如何沟通？

在一个效率低下的环境中，人们必须至少在管理层次体系上向上请示一层才能与他们同一层次的不同团队或个人沟通。层次越多，组织结构越庞大，这个过程效率就越低，至少需要牵涉更多的人。

### 决策过程是否需要一个正式会议？

如果做出更改需要某种书面工作，倘若这个系统中很多工作仍需要手动介入而没有实现自动化，这也会对灵活性和生产力有负面影响。

### 要完成一个更改需要在管理层次体系向上请示多少层？

如果一个人需要完成某个更改，尽管这个更改只影响他们团队自己的工作，但仍然需要得到其直接经理以上的多个管理层的认可才能做出更改，这会让他们感觉很受束缚。

## 案例研究：政府数字服务

在下面的案例研究中，我们会介绍某国的政府数字服务小组（Government Digital Service, GDS），这是某国政府内阁办公室下设的一个团队，这个团队负责完成政府数字服务的转换。<sup>注9</sup>2010年，Martha Lane Fox整理了一个名为《Directgov 2010 and Beyond: Revolution Not Evolution》的报告。GDS成立于2011年4月，由Public Expenditure Executive (Efficiency& Reform)监管。

---

注9： 这个案例研究中的所有观点都是我们对所提供信息的解读，不代表某国政府的观点。

## 显性文化

政府数字服务建立了7个数字原则：

- 默认数字化。
- 用户为先。
- 从过程中学习。
- 建立信任网络。
- 移除障碍。
- 为技术领导人发展创建环境。
- 不要每一件事都自己做（也做不到）。

从用户开始不只是我们建立技术的方式，这也是我们建立政府的方式。

——Jennifer Pahlka, Code for America Summit

在这里我们可以看到，这些原则中明确地反映了价值观和禁忌。例如，“用户为先”是一种价值观，指出用户比其他方面（比如说，一个工程师想要试验和学习一种新工具或架构）有更高的优先级，“不要每一件事都自己做”则是一种禁忌。需要指出很重要的一点，这里正面指出的价值观要多于负面的禁忌。尽管告诉人们什么不要做可能更容易，但是如果能明确描述你想要的行为，这在创建期望的文化或氛围时可能更有效。

结合这些数字原则，GDS还建立了10个设计原则：

- 从需求开始。
- 少做。
- 用数据来设计。
- 做难的工作使它变得简单。
- 迭代。然后再迭代。
- 这适用于所有人。

- 理解上下文。
- 构建数字服务，而不是网站。
- 保持一致，而不是统一。
- 保持开放：这会让情况更好。

这些设计原则进一步体现了前面列出的一个数字原则：用户为先（在这里，就是强调用户体验的所有方面，而不是后端代码）。在他们的文化中，最重要的思想是这不只是硬件和软件技术的转换；转换也会改变用户的体验。Claim Carer的Allowance数字服务实现就是这样一个例子。



通过建立显性文化而不是依赖于隐性的理解（这通常会导致误解），GDS明确地强调了希望在他们的政府领域完成哪些类型的改变。

## 计划

计划是所有关注软件开发（或数字服务）的组织的一个重要部分。通过明确地指出他们希望做什么以及何时完成，可以确定给定时间范围内他们优先的工作，不仅如此，团队也更有可能达到他们的目标。这与有一个显性文化紧密相关，如果不能很好地定义目标来做出计划，就不太可能达到目标。

对于要完成的更改，GDS团队的计划过程包括花足够的时间检查可能的解决方案以及所提供方案的价值，确保选择最能满足其需求的解决方案，同时还要满足其数字和设计原则。他们还与政府的其他团队交流来进行协调，确保没有其他团队在做他们计划做的工作。这个步骤听上去很简单，但非常重要，这可以确保所有人意见统一，尽可能有效地工作，而不会浪费时间重复劳动。

一旦收集到项目及其需求的有关数据，他们会评估可能的解决方案，包括开源和商业解决方案。他们创建了一个原型，并向多个团队共享这个原型，包括开发、运维和服务经理团队。这使得他们在最终提交解决方案之前可以从尽可能多的重要利益相关人那里得到反馈，同样地，这样可以尽可能减少最后因改变方向而带来的浪费，而且可以确保候选解决方案能够尽量满足所有人的需求。





在一个组织中（特别是大型组织）计划项目或其他工作时，要注意：

- 有没有其他团队在做这方面的工作？
- 如何协调和结合其他团队的工作？
- 需要涉及哪些利益相关人和决策者？
- 如何定义这个项目的成功？

## 挑战

政府机构面临的挑战之一是不同团队经常要做重复的工作（见图14-3）。每个团队除了它们特定的重点工作外，都必须为其应用提供核心基础设施元素。由于这些核心服务不是集中的，每个团队都必须花额外的时间来聘请有这方面技能的专家。

可以在政府中创建多元化平台服务，通过提供一个集中支持的服务，同时满足法律要求的所有安全和隐私约束，可以节省服务团队的时间。不再需要每个团队自己提供和维护满足其需求的服务，而是集中地提供服务，这就允许服务团队重点关注他们的特定领域、技能和需求。政府中这种多元化平台的关键需求包括：

- 应当是自服务。
- 必须采用多个服务提供商。
- 代码、数据和日志必须隔离。

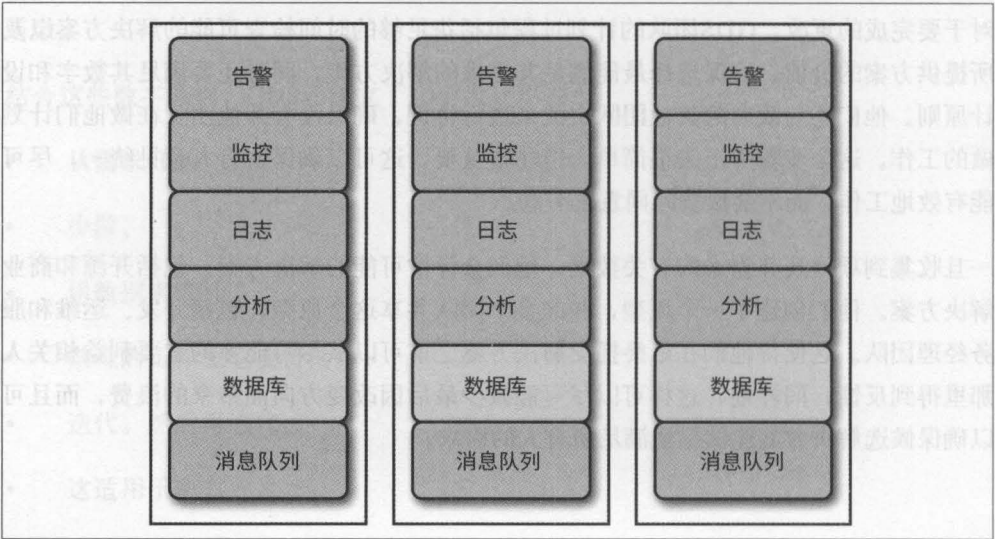


图14-3：政府中重复的服务

自服务应用使人们可以完全控制所需要的应用的各个方面，允许平台团队把重点放在平台的改善上，而不是如何得到他们需要的服务。这有利于服务和平台团队的专业化。

第二个需求是要有多个服务提供商，这可以避免开发商锁定，鼓励竞争定价，并消除单一故障点。在研究项目的需求时，体现出使用多个公共云而不只是某一个云的重要性。提前了解这个需求，就能避免使用开发商特定的特性，否则有可能导致锁定，以后要改变提供商或者扩展为包括更多提供商时会更困难。不过，后来，GPS决定不再有这个需求，因为利用开源解决方案可以为他们提供同样的灵活性。

对多元化平台的第三个需求（也是最基本的需求）是机构之间代码、数据和日志的完全隔离。这对于确保各个团体或机构满足其安全和隐私约束非常必要，如果不能保证这一点，这个平台就没有用，即使另外两个需求得到满足也不能认为是成功的。这个挑战的细节可能是一个政府机构特有的，很多组织还有各种必须满足的法律需求（如PCI, SOX或HIPAA合规性），所有这些都应当从一开始就作为计划过程的一部分，以避免浪费人力物力。

## 建立亲密性

GDS建立亲密性的一种方法是参加Global GovJams。GovJam类似于一个编程马拉松，参加者要在有限的时间内（在这里就是48小时）完成他们的Jam项目。受Global Service Jams和Sustainability Jams的启发，GovJams不限定主题，由人们对想法投票，并创建关注公共事业的团队。

一旦建立团队，他们会与委托人交流，找出他们的需求。不同于传统的编程马拉松，Jam强调对改进和创新感兴趣的人之间的合作。Global GovJam通过一个公共的主题和共同的集中平台将世界各地的人联合和联系在一起建立原型。另外，参加者在Twitter上用#ggovjam协作和分享。

然后团队使用已有的资源以及他们在48小时时间窗口内利用集体智慧完成的工作来建立原型。他们会定期演示，观察其他人如何使用这个产品，根据他们学习的结果改进产品。团队之间会分享想法并提供帮助，因为大家的关注点是要有益于他们的委托人，而不是他们自己赢得某个奖励。由于强调协作，这也非常有利于将这种协作思维带回GDS。

GDS建立亲密性的第二种方法是定期发博客来分享关于组织、目标和所完成项目的信息。尽管仍然要求符合各种安全和隐私法规，不过也鼓励GDS员工写文章在团队的公

开博客上发表。他们的博客文章涉及各种主题，从委托人可用的新产品或服务到他们改进文化和流程的不同方法都有涉及。这不仅使团队有一个亲切的形象，还有助于建立学习和分享的习惯。

最后，GDS还通过参与开源社区来建立亲密性。政府数字服务建立在开源软件基础上。通过使用开源软件，可以帮助技术人员关注用户需求。他们在<https://github.com/gds-operations>和<https://github.com/alphagov>提供了他们的代码。

James Stewart是GDS技术架构主管，通过引述JP Rangaswami的一段话描述了选择工具的一般方法，如图14-4所示：

对于常见问题，用开源（Opensource）。

对于少见问题，要购买（Buy）。

对于特有问题，要构建（Build）。

基本说来，这体现了团队中每一个人都要使用开源解决日常问题，通过解决对社区整体有影响的问题为之做出贡献。对开源做贡献并不只是要提交代码以及提供文档、bug报告和示例等等。对于很少见的问题，要购买产品来解决。最后，对于你的组织或团队特有的问题，构建你自己的解决方案比较有效。

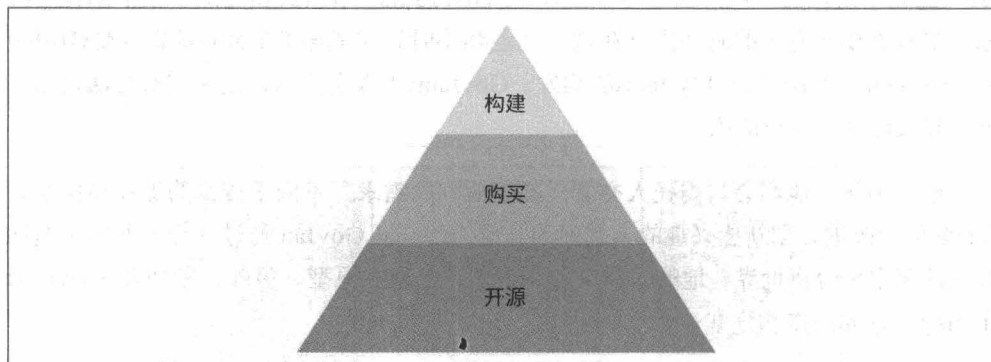


图14-4：构建、购买和开源金字塔

总的来说，政府数字服务团队在努力创建和维护一个明确定义的文化，非常重视用户以及解决用户面对的问题。除了代码，还包括设计和总的体验。另外强调使用敏捷实践来减少浪费、更快地迭代，并帮助为政府其他部门简化数字和技术服务。他们不仅关注协作和亲密性，而且有自己的价值观和禁忌，这些对于创建政府所需的文化很有帮助。



# 案例研究：Target

在过去的10年里，我们已经看到人们的购物方式、地点和时间发生了巨大的变化。Target顾客需要随时随地都能得到可靠的服务，这个公司之所以能够针对供应链的演进格局和客户需求快速调整战略，它的底层技术至关重要。

由于Target是一个很大的组织，采用devops文化不是一个自顶向下的单方面的决定。多个团队分别开始了他们的devops旅程，然后联合展示了选择这些策略取得的成功。这个案例的信息是通过公开的博客文章、Target员工公开的演示稿和公司宣传单收集的。

## 分析Target

Target公司是一家总部位于美国明尼苏达州明尼阿波利斯的零售公司。最早由George Dayton建于1902年，当时名叫Dayton Dry Goods，前面提到过，2015年中期Target有大约347000个员工。他们是2015年美国第二大进口商。除了Target实体店，还扩展到银行、医药和保健行业。

作为一家现代零售公司，Target有很长的技术革新历史。1988年，Target在所有Target商店和配送中心使用了UPC扫描器，这改变了顾客的店内体验，排队队伍更短，商品在货架上的摆放也更合理。这项革新经过多年的发展，已经提供了从内到外的各种支持技术，包括比价设备、收银机、检查库存的手持设备、礼品部、确保配送中心适时适量存货的应用等。

## 从期望的结果开始

在一个发展历史很长的大型组织中，要完成改变，这个旅程通常就像是有很多急转变的复杂迷宫。大家所走的道路上包括一些能减少风险的最佳实践。这个组织充分体现了长时间发展所带来的复杂性。如果从更高的角度来看，可能很难为Target的devops之旅定义一个特定的起点。

实际上，确实没有唯一的起点。有多个团队从不同的起点起步。Heather Mickman领导API和集成开发团队，Ross Clanton领导运维基础设施服务团队，这两个团队都在组织中努力实现devops。各个团队的道路都不容易，有些团队开始时甚至没有使用devops这个词。实际上，他们只是描述了期望的结果，如体现devops哲学的“更简洁、更快、更高质量的服务交付”。Heather Mickman解释说：

我必须停止使用devops这个术语。这已经变成一个过于泛滥的词，有太多的误解和误区。我和同伴以及组织中的其他高级领导人交谈时，我清楚地看到，当我说到“devops”时他们都不说话了。

Mickman决定停止使用devops一词是因为它变得过于泛滥，这也体现了第2章介绍的大众模式的强大力量。在你自己的组织中也可能会看到类似的反应，人们的思路会发散和转移，而不能有效地讨论。正如Mickman发现的，如果你把关注点放在期望的结果和改变上，而不要过分关注术语，就能在组织中更容易地实现改变。改变是分阶段的，开始是个人的改变，经过众人努力、从上到下的支持，然后扩展成功的策略。



要关注期望的结果，并注意语言的敏感性，这对于完成改变非常重要。可以讲一些吸引人的故事引起和推动改变，而这并不需要使用devops这个词。了解Target当前格局很重要，可以了解现有工作对人们有什么影响。

## 企业中的亲密性

Target领导层提倡现有的组织性学习文化，以此鼓励在企业中建立亲密性。通过采纳组织性学习的思想，并鼓励每个人提升，这使他们可以将改变扩展到整个组织。他们强调的4个要素包括：

- 试验。
- 测试。
- 失败。
- 成功。

不同团队的大量经验使Ross Clanton对各个团队的难题以及组织中不合适的激励、转手和当责有了自己的看法和理解。

Clanton在Target的devops之旅就是从寻求指导来帮助解决这些难题开始的。很多人推荐 Kevin Behr, Gene Kim和George Spafford写的《The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business win》。读过《The Phoenix Project》之后，Clanton多买了几本，发给团队中的其他人，甚至举办了一个场外活动来演习这本书中的不同场景。

他与组织中的其他技术领导人（包括主管Heather Mickman和工程领导人Jeff Einhorn）组队，在整个组织中寻找合作伙伴来一起推动所需的改变。这对于帮助弥合各个孤岛之间的缺口至关重要，不同的组织有不同的优先工作和需要克服的挑战。他们找到这个领域中的其他专家，包括Netflix、Google和Facebook的领导人，来了解可以在他们自己的组织中采用的适用模式。



要指出重要的一点，他们不仅在组织内部推进学习和推动改变，另外还在外部寻找灵感。如果你只在自己的组织内部寻找想法和策略，可能会有这样一种风险，最后会得到你原来已经有的同样的实践、习惯和模式，至少如果你想做出重大的组织改变，其中的一些策略可能并适用。

不要陷入货船崇拜，只是因为其他人在改变而你想复制他们的成功就做同样的事情，另外不要低估外部经验和技术的的重要性。

在2014年的DevOps企业峰会上，Ross Clanton将他在Target的devops之旅描述为在整个组织中将人们联系起来。在有限的团队中获得成功之后，他们开始以多种方式推广到Target的更大范围：

- 2014年2月，他们与特邀发言人Rob Cummings（Nordstrom）和Michael Ducy（Chef）召开了第一次内部devops微会议。
- 他们定期举办活动，邀请其他外部特邀发言人，包括Jeff Sussna、Fletcher Nichol、Ian Malpass、Sean O'Neil和Jez Humble。这些内部活动有力地推动了这项非正式运动不断成长，第一次活动有160个参会者，而到2015年2月的活动中参会者已经超过400人。
- 为了鼓励希望更多地了解这些主题的人，他们启动一个每周自动化开放实验室项目（Automation Open Labs program），这是一个开放式的Q&A研讨。
- 另外他们还每月举办一次演示研讨，为社区成员提供一个机会来分享他们的工作、给出反馈、启发和得到启发。

他们通过融合精益、devops和敏捷运动来培养跨学科教练，使“教练”在组织中采用一致的方式传达信息；另外通过高度沉浸的指导课程鼓励精细质量实践方法；还鼓励员工在<http://target.github.io>上分享他们的经验，所有这些方法不仅增加了组织内部的亲密性，同时也增加与外部的亲密性。



## 企业中的工具和技术

Mickman团队的任务是构建应用程序接口（API）来简化Target系统和组织孤岛不断增长的复杂性。

在一个采用瀑布方法的组织中，一段时间后会有一种趋势，会趋向于最小化风险的层次结构和流程。角色变得严格，团队专注于一项服务，这会与组织中的其他服务紧耦合。这将导致更长的发布周期，要在组织中对可能的变更进行沟通，而这又会带来更大的风险。

API开发被看作是处理组织文化和技术负债增长的一种方法。作为一家零售公司，Target开发了一些重要的API，如商品、商店位置和开放时间、促销和定价API。这些API不断演进，使得Target可以快速测试和学习如何优化客户体验以及Pinterest等合作伙伴的体验。

### devops安全的重要性

2015年12月，Target的心愿单应用被报告存在个人数据泄漏问题。API的关键是为软件提供一个通信接口来实现交互。在开发软件时，通常公司会关注这个软件要如何使用，而没有关注用户可能会如何使用。

创建软件来支持软件相互通信时，实际上也是在支持人与软件通信。通常会设置授权和认证措施来检验谁在通信和使用软件。认证确保一个人确实是他声称的那个人。授权则定义了有关一个人能够做什么的访问策略。

关于Target的这个漏洞，这个API看起来没有提供充分的授权或认证。为一个服务增加认证的过程可能很困难，特别是如果允许人们根据已知信息查找一个心愿单（而不要求创建用户基本信息），这会更困难。尽管创建心愿单的人希望向组织分享他们的标识信息，但读心愿单的人可能并不愿意这么做。所以不在心愿单上设置认证也是有道理的，可能不算一个糟糕的决定。

但是这就带来一个挑战，他们不仅要创建一个安全的服务，不能向不认识的人泄漏个人的标识信息（personal identifying information, PII），同时还要提供一种方法来搜索PII查找匹配的人的心愿单。这个API的关键缺陷是缺少一组明确的实体访问策略。未认证、未授权的实体不能访问PII，即使他有一个用户ID。理想情况下，用户ID不能很容易猜出。对于Target的这种情况，在有些方面，看起来如果有一个人的某种信息，就可以解释为已授权可以得到他的所有信息。

这里反映出有一点很重要，要完成转换，组织中的每个部分都应当参与转换。在累积了大量文化和技术负债的组织中，随着长期存在的孤岛缓慢地连接起来，我们会不断看到这种影响。

对于Target领导层，重要的是要找到适当的工具，从而得到他们想要看到的结果。正如第四部分中所讨论的，究竟是特定的工具重要还是使用工具的方法重要，个人以及组织对此往往有不同的看法。Target的观点是这些工具确实重要，因为适当地选择工具可以使他们拥有“全栈的”平台和API，而不会让它们分离和割据。

为了实现这一点，他们结合使用OpenStack平台来完成云计算，使用Jenkins提供持续集成和持续交付，使用Chef提供基础设施自动化或配置管理，还使用GitHub提供源控制。这种组合对他们尤其有意义，因为这允许开发过程中有更大的透明度。不再需要开发人员自己完成单独的离线代码审查，现在他们使用GitHub拉取请求，运维人员非常欢迎这一点，因为现在他们可以了解发生了哪些改变，而且可以更容易地参与到开发过程中。

在此基础上，他们发现使用Hipchat来进行内部沟通很有好处，chatops越来越成为他们开发过程中不可缺少的一部分。利用Hipchat，他们发现不仅团队内部更容易地沟通，还有助于与其他团队沟通。由于他们使用chatops在适当的通道发布警报和事件，可以实时掌握开发状态，这使他们能够比以前更快地对事件做出响应。

Target通过API数目的增长来度量这个新工具项目的成功，2014年10月的API个数为30个，到2015年2月已经增加到45个。API的个数很重要，因为这允许团队和服务集成和协作，而不是互为孤岛，彼此割据。不仅内部和外部API请求数量增长，而且即使网站保持稳定（每月事故数很低），API请求仍在增加。通过找到适当的工具组合，可以帮助他们更多地协作，同时消除之前存在的一些痛点问题，这是他们取得成功的一个重要部分。



要定义在你的组织中成功使用工具的含义。应当关注你想要得到的结果，明确你的痛点问题是什么，然后找出解决这些问题的工具，而不是反过来寻找能够用某个特定工具或技术解决的问题。

## 在企业中分享知识

在有限的团队中获得成功之后，Mickman和Clanton开始把它推广到Target中更大的范围。推动整个组织完成这个改变的一种机制是每季度召开内部devopsdays会议，他们

还从其他组织请来特邀发言人做主题演讲，包括Nordstrom的Rob Cummings和Chef的Michael Ducey。前面已经提到过，这不仅有助于引入新的想法，还会使大家对他们自己的devops计划和进展更有兴趣。

他们采用的第二种策略是确保整个组织有一致的方式传达信息。他们提出了教练的想法，即精益、devops和敏捷等领域的主题问题专家帮助指导其他人或团队讲授和分享信息。他们发现，通过将这些主题融合为一个主题，并通过培养跨学科教练，他们能够在内部建立一种更一致的大众模式，使得在讨论不同的devops相关概念时，更多的人能建立共识。

在整个组织开办沉浸式指导课程也是扩展这个计划的一种方法，这种方法在多个团队中效果都很好。整个组织的人都可以参加开放实验室，这会让更多的人能够讲授和分享知识，“自动化编程马拉松”也给更多的人提供了一个参与这些改变的机会，并且可以更好地完成他们自己的工作。

Target采用了很多方法来找出对他们适用的工具和协作解决方案，以及分享在这个组织中有效（和不可行）的策略，以上只是其中的部分方法。由于一方面得到了从上到下的支持，另一方面人们在个人和团队层次上为推动改变做了大量努力，这些共同促成了Target的成功。



实际上，如果一个组织自信地认为在做正确的改变，这种方法就很有必要，即让个人贡献者和团队根据他们自己深入的经验推动改变，不过管理层的支持也是必要的，这会促成改变的真正完成，而不是无法进行下去。

## 小结

基于我们过去的决定以及当前的目标，会在我们的环境中会遇到一些挑战。如果我们从个人、团队和组织层次来解决这些问题，可以成功地评估、规划和克服这些问题。实际上，从本质上讲，采用devops的关键挑战就是规模化问题。不同层次上的障碍会由于历史而放大，这会影

响我们如何应对组织生命周期中出现的不同拐点。有时我们需要重新评估当前流程并反思。有时需要慢下来谨慎考虑；有时则需要快速做出调整。只有不断的实践并从错误中学习才能最终获得成功。

成功地规模化是一门艺术和科学，要认识到何时以及如何根据需要进行改变，来适应不断变化的环境。就像户外攀登一样，你的特定环境中没有已经做好彩色标记的路线



来为你指引道路。要提高必要的个人、团队和组织技能，这会让你做好准备，不论组织的规模和复杂性如何，有效实现devops的基本原则都同样适用。

# 规模化：误区和问题排查

下面来分析个人可能会遇到的一些常见的规模化挑战，以及如何排查。这一章将介绍与管理者和个人贡献者有关的方面，你可能发现直接适用于你的当前角色的内容更有帮助，不过，了解其他角色面临的挑战也很有意义。

## 规模化误区

上一章中我们讨论过，规模化不只是成为一个非常庞大的组织，或者完成一些单独的“企业devops”。

### 有些团队永远也无法合作

通常会把开发和运维团队的不一致描述为devops的起源。一般地，团队可能由于误解陷入一种高度紧张和冲突的状态。如第二部分中提到的，冲突不是坏事。对于一个健康的组织，这是一个很重要的信号。要确保团队相互之间没有不一致，这一点至关重要。通过个人调解，可以使个人之间达成共识，而团队调解需要一种更复杂的过程。修复团队内部关系很重要，另外还需要修正导致失败的流程。

修复的初始阶段通常都很困难，团队要重新了解如何交互，这可能很费劲。需要时间并积累正面经验才能重建团队之间的信任。如果有一个负面经验，这会破坏已经取得的进展。

需要注意以下行为模式：

**批评 (Criticism)**

对性格特征的个人攻击。

**轻视 (Contempt)**

来自优越地位的言论或行为。

**防卫 (Defensiveness)**

自我保护的言论或行为。

**抵制 (Stonewalling)**

有情绪地拒绝交互。

**地毯式攻击 (Blanket accusations)**

包括“他们总是……”或者“他们从来不……”（或者“我们总是/从来不……”）的言论。

如果团队采用一种“我们”而不是“他们”模式，这会导致同一性出现问题。在一个组织中，团队要对愿景和成功的定义达成一致，这至关重要。如果一个团队感觉他们的目标会受另一个团队的影响，缺乏后者的信息和透明度会影响其目标的实现，他们就可能带情绪地对第二个团队施加压力。这可能会加剧团队之间的紧张气氛。

对于分布式的团队，这种情况会更为严重。如果有不同的优先工作或成功目标，冲突会加剧，需要花额外的精力来解决。对于分布式团队，要保证这个解决冲突的过程对所有人都是公平的，使他们相信自己也参与了这个解决过程，这很重要。

研究表明，现场参观可以帮助同事建立密切的关系，增强合作，还能帮助人们对沟通和工作的方式建立信任和理解。<sup>注1</sup>对于何时以及如何参观，要尽可能为人们提供选择，而且在团队合作的整个过程中要保持透明。如果人们完全不了解会对他们有直接影响的决策，即使他们认可这些决策的结果，也会感觉很不好。

两个组织或公司合并时，通常也可能导致两个团队出现不一致，以至于影响他们的工作。要尽可能确保两个团队都有利益相关人在场和参与，这很重要。这有助于减少“我们比他们强”模式的滋生。

---

注1： Pamela J. Hinds和Catherine Durnell Cramton, “Situated Coworker Familiarity: How Site Visits Transform Relationships Among Distributed Workers,” *Organization Science* 25, no. 3 (2014)。



有些人会为自己构筑一个“自给自足”的小岛，将知识占为己有不与人分享，以此保证自己不会失去工作，他们也会有这种心态。这些人喜欢作为单一故障点或知识传递的瓶颈，因为他们相信这会体现他们的价值，使得团队或组织少不了他们。这会与他们与同伴之间的关系产生负面影响，应当采取措施传播他们的知识并修复这些关系，包括这些人和所在组织之间的不信任。

## 我们需要得到高层的完全支持才能开始改变

作为一个管理者，你对你的直接下属和他们的效率会有很大影响。如果高层管理层推诿，你要作一个“保护伞”，<sup>注2</sup>也就是说，保护你的下属，尽量不要影响他们的正常工作。

如果人们对你的团队有不切实际的需求，你可以保护你的团队，使他们不分心，或者简单地将这些要求传给他们。

尽你所能地加强你的团队，尽量给他们足够的空间来试验工具、工作流和实践。取决于你的组织文化，你会发现用实际结果来说服高层会更容易，例如展示一个试验的结果，表明一个新工具或工作流比现在的工具或工作流更有效。要尽力保护你的团队免受干扰，同时允许他们确定对他们最适用的方法。

## 我们现在没有招聘预算，所以不能开始devops

在规模较小或快速发展的组织中，转变为一种更强调devops的文化可能很简单，只需要确保聘用具备这些技能和思维模式的人，但是并不是每一个团队或组织都可以大量地聘用 人员，或者甚至根本不能聘用。好在，即使不聘用一个由“10倍明星devops工程师”组成的全新团队也完全可以实现devops。

### 建立对目标的共同理解

为了让人们对devops之类的想法达成一致，首先需要确保每个人对这些想法的含义达成共识。要能够对你希望你的环境中做哪些改变以及为什么要做这些改变进行沟通。特别是在可能重组的较大环境中，人们会对听上去很轻松或时髦的改变表示怀疑，所以要说明你希望的具体改变，以及你希望从这些改变得到哪些具体的好处。

---

注2：这个词是Todd Jackson提出的，他是Google Gmail的前产品经理，他说，作为一个管理者，“你可以是一个倾听者或者保护伞”。

## 提供和鼓励学习机会

我们在第8章讨论过，让一个老系统管理员学习新技术是完全可以的。要实现devops变革很可能需要学习新的软技能和技术技能，例如如何促进无问责事后分析，以及如何使用Docker。如果你希望你的员工具备某些技能，就要确保提供相应的培训，而且要鼓励整个组织有一个学习环境和成长型思维模式。

## 创建一个坚持devops原则的环境

为了保证这些改变是持续的，需要创建一个人们乐于建立的环境。如果你希望人们能更多地沟通或者指导，要在你的技能矩阵或职业发展计划中明确指定这些技能。如果你反对粗鲁的行为，要鼓励人们一旦看到这种行为就制止（如果他们愿意），并确保违规或恶意行为会有相应的后果。很难有协作和“不受欢迎的人”并存的环境，所以如果你能容忍这样一些家伙，就说明你没有积极建立一个协作的devops环境。

如果想改变一个现有环境以及这个环境中人们的习惯和行为，可以先从小的步骤开始。确保提供清晰的目标、持续的学习机会，以及一种信任、理解和协作的氛围。

# 规模化问题排查

没有一种十全十美的解决方案能够为每一个组织的规模化问题提供全部答案。这一节将介绍组织在其生命周期过程中发展变化时出现的一些常见情况。

## 管理层坚持让我们继续使用X，没有看到devops的价值

我们在上一章讨论Target时已经看到，在某种程度上，必须有管理层的支持才能使这些改变持续地影响整个组织。不过，他们也看到，并不是从一开始就能得到认可。可以从一个团队开始，给这个团队一些时间来进行试验，一旦有可以展示的正面结果，就向管理层报告这些改变，这是一种很有用的方法，这样就能向其他人展示出这些改变的价值（而且立竿见影）。

如果管理层还有问题，看看能不能让你的管理者参与推进你希望的改变。他们可能：

- 避开现有的限制。
- 代表你与其他管理者协商。
- 允许你在你的团队中试验。
- 为你屏蔽所有质疑。

如果你的直接管理者没有看到devops的价值，很自然地，要完成改变会更为困难，但是还是能够以同样的方式在团队中寻找同盟。如果你的管理者坚持使用某个工具或某个方法，有没有办法可以同时使用一个新工具或方法Y，看看一段时间后这二者的比较结果？仔细研究“管理你的管理者”之类的想法，看看是否有办法更有效地沟通，让你的管理者了解为什么你希望做出改变，并使他们也能感受到好处。

## 团队过于分散

如果团队总是无法满足他们的需求或最后期限，而且你确定对他们的要求并没有过于不切实际，那么原因很可能是给定的任务或项目没有足够的人手来满足这些需求。应当花一些时间重新评估项目的工作负荷和最后期限；你可能发现需要在项目之间调整人员，或者针对当前团队能完成的目标调整最后期限。

你可能发现需要聘用更多的人，特别是在成长阶段，另外还要当心衰退阶段，在这个阶段你可能要减少人数而不缩减项目范围或工作负荷。很可能要让更少的人来完成同样的工作量以削减成本，但是如果超过了某个临界点，这只会带来低质量的工作和倦怠的员工。要保证你对人们的期望切合实际，这很重要。

要想让同样多的人有效地完成更多的工作，一种办法是支持他们采用更聪明的方式工作，而不是更努力地工作。不是要工作更长时间来完成更多工作，应当考察当前的工具和流程，查看是否有改进的办法。与你的团队成员讨论他们的工作流以及他们使用什么工具和技术来完成工作，找出更高效的工作方法，从而减少时间和精力浪费。你可能需要在组织之外找想法，不过也要结合当前团队成员的建议。如果人们出于习惯不愿意给出他们的想法或提出建议，可能要多花一些时间和努力，不过最终你会从结果看到一些很有价值的信息。

## 决策不当

要认识到决策经常会带来不想要的结果（“我们决定尝试X，但最后它让情况更糟，而不是更好”），或者决策过程看起来要浪费大量时间和精力，这是组织成长变化时一个常见的痛点问题。组织规模很小时，例如你的整个工程部门都在同一个房间，可能很容易做出决策，而当组织发展为包括遍及世界各地的多个部门时，原本很容易做出的决策会变得迟迟不定。可以用以下方法来解决这个问题：

### 检查流程

随着组织的成长，关于谁有哪些问题会越来越混乱。当问题或项目涉及多个团队时，你可能会发现这些问题要么被草率略过，没有人“认领”，要么多个团队都



认为问题是自己的，因此相互竞争而导致冲突。后一种情况尤其麻烦，这会导致决策过程效率低下，因为有太多的争论，以至于很难做出任何决策。另一方面，自上而下的决策尽管能大大减少争论，但可能会使人们对这个过程不满，也可能对结果不满意。

### 认识清晰度问题

如果不清楚需要做什么决定或者这些决策有什么影响，或者如果结果有太多未知因素，就有可能导致分析瘫痪而处于停滞状态。

### 注意过度的流程

如果有太多流程，以至于流程本身会妨碍工作或者导致无法在合理的时间内达到某个结果，人们就会避免做出决策。

### 衡量生产力与风险

如果人们避免做出决策，这会影响生产力。花些时间来了解做错误决策的风险。如果做错误决策影响很小，继续犹豫而不做决定就是浪费时间，并影响决策的有用性。

### 增量改变

要认识到大多数决定并不是不可逆的，这会帮助改变我们的认知框架。如果我们认识到可能在事后改变想法，这会消除因为有太多选择而产生的分析瘫痪。

### 建立试验的安全空间

错误会教我们如何适当地分析风险以及在需要时做出正确的决策。这个领域中无问责文化（而不是问责文化）会带来很大好处，因为人们会更关注得到最好的结果，而不是相互指责。



跟踪你做出的决策。如果继续做同样的事情而希望看到不同的结果，这是不理智的，只会浪费时间和精力。通过跟踪决策和结果，你可以做出修正，并对所做的决策更有信心。

## 在吸引我们想要的人才方面有问题

确保对人力资源部门说明职位需求，参加当地研讨组，重新考虑职位需求，并反馈社区。有些环境中，如果你的组织的历史会影响有多少人来应聘或者哪些类型的人来应聘，可以采取一些额外的措施，这可能会有帮助。

首先确认环境的历史和挑战。人们可能会对报告的一些情况提出一些问题，要准备好回答这些问题。如果一个人对你的公司做了研究，还读过Glassdoor上的评论，而且仍然来参加面试，这很好（说明他对你的公司确实感兴趣）！要鼓励你的员工通过演示或者参加大会讨论他们的工作。演示和参与的过程应当透明而明确。

找出问题是否出在招聘过程上，或者面试是否有问题。建立一个面试培训程序，另外可以结对面试，使组织中更多的人成为更好的面试官。明确你的某些面试手段是否会让潜在的应聘者却步。如果面试为时全天或持续多天，或者面试官态度过于敌对，都会让应聘者不愿意应聘或接受这个职位，而且这也不利于发展组织的多样性。

与之相关，也可能你的组织中有些人在行为方面有问题。如果这些问题长时间没有解决，可能很难表现出来，因为人们会认为这些有问题的行为可以接受，因此不再打算报告或解决这个问题。一定要从组织中清除这些有问题的人，这对于吸引更多的应聘者是必要的，特别是如果这些有问题的行为涉及种族歧视、性别歧视或者类似的差别对待。

要了解你可能聘用的新员工，最大程度地利用你的候选人才库来实现这一点，这也很有帮助。如果你是一个大型企业组织，一直以来都对行业变化反应很慢，就不要去“挖”初创公司的员工。要注意你的组织存在的问题和限制，这会帮助你寻找适当的候选人，并避免得到一个乱招聘的坏名声。

## 重组或裁员后士气受挫

一个组织生命周期中会有衰落阶段，这是一个很自然的阶段，在这个阶段领导层可能会减少生产线或员工人数。不断改变的文化规范会增加透明度，因为人们可以对体验、工资和面试信息发布评论。Glassdoor就是这样一个网站，员工可以匿名地发布这些信息，而未来员工可以阅读这些信息，来确定对某个职位有多大兴趣。

进入这个阶段时，公司所采用的流程将反映它的文化。如果文化的内部认知与其发布消息的含义不一致，领导层以何种方式以及何时传达要完成的改变可能导致认知矛盾。

如果对不同的人评价方式不同，也会对员工的士气产生负面影响。如果一个公司发布消息称绩效差的人将被裁员，却将一个很出色的团队成员辞退，团队中其他成员就必须（做一些猜测）建立缺少的上下文来处理这种认知不一致。

如果改变没有得到很好的处理，这会让留在公司的人士气低落；组织中个人和团队之间的合作、协作以及团队工作会减少；由于压力和疾病导致事故和旷工增加。取决于重组或裁员的原因，这可能会导致与期望相反的结果。

临近减员时，不要试图隐藏。组织越大，消息泄漏的机会就越多，这可能是人们对此很关心并希望能调整自己的情绪，也可能因为有人错误地决定分享敏感信息。如果你通知公司之前这个新闻先行公开，你会让人们有很大压力，认为他们会失去工作，或者会失去优秀的同事。

如果你的公司有过这种情况，而且处理不当，由于分享经验的透明度不断增加，一定要抓住机会解决其余的所有内部问题，这至关重要。

审查和修正你的招聘方法。确保你的招聘方法给人一种正面的体验。即使有不好的评论，如果对电话约谈和面试过程有很好的体验，这也能帮助修复你的形象。

寻找入门级和初级应聘者。对他们进行指导。尽管这些人在短期内可能有更大的花费，不过从长期来看对你会很有帮助。他们会带来不同的视角，使你在招聘中付出的努力得到回报。

最后，开除捣乱的人。跟踪和度量个人对团队的影响。不要因为某个人确实很聪明或效率高就为他的糟糕行为找借口。他们的行为会影响团队的其他人，而且可能会影响来面试的人。

## 我们不知道是否需要为X建立一个完整的团队

即使一个项目或角色的日常工作不够多，好像没有必要聘用多个人，但单人团队可能会导致倦怠和单一故障点。“公车因子”是一个数，描述一个团队或项目失去多少个人就会失去继续前进所需的组织知识或能力（或者多少人生病了，项目/团队就失败了）。这个数字越低，就越糟糕，因为这表示对错误的回旋余地越小。

根据定义，单人团队的公车因子数为1：如果这个人（没那么糟糕）只是生病了，想要休假，或者要离开公司，会怎么样呢？即使没有足够的工作量需要聘用一个完整的团队，起码要寻找分享知识的方法。结对和保证文档更新就可以作为很好的起点。

倦怠是单人团队存在的另一个严重问题。如果只有这一个人知道如何做某些事情，他们身上可能会有很大压力，可能来自自身，也可能是外部施加的压力，这会让他们不能考虑请假或者不能生病。他们没有时间充电，压力会不断上升。最后也更有可能会离开这个组织，可能因为他们主动想要找一个没有那么大压力的工作（不再作为单一故





# 搭建DevOps文化桥梁

## 4大支柱搭建桥梁

devops要保持有力而且影响长远，关键要素之一是它的灵活性。正如我们在本书中展示的那样，没有一种实现devops的特定方法，它不需要某个特定的软件或流程，也不仅限于Web创业公司。

确实有一些故事（Netflix和 Etsy）可以成为成功实现devops的范例，但是对于组织可以采用哪些方法使用有效实现devops的4个支柱提高他们的生产力，这些故事并不是无所不包的。有些组织（如Etsy）因其文化和技术实践而闻名，肯定有一些可以分享的重要故事。我们有意扩大了故事的范围，而不仅仅限于devops圈子中常留遗憾的那些故事。这些故事的多样性并不影响devops的重要性，而是证明DevOps对于我们的工作方式以及整个行业重要性的关键。

我们在讲关于devops的故事时，通常会谈到孤岛。一个常见说法是开发和运维团队就是孤岛，相互割裂。他们几乎从不沟通，更不要说有彼此协作。人们希望找出办法来消除这些孤岛。我们更喜欢从一种更有建设性的角度去感受devops，而不是一种破除障碍的说法。

并不是将作为孤岛，我们只是把不同的团队割裂开看作孤岛。我们使用一系列跨主机架构的生态系统，多个站点的部署能随时发生。通过测试，部署可以任意、快速地移动。因此，我们需要在这些孤岛之间搭建桥梁，而不是消除它们。我们的目标是构建健壮，利用有效实现devops的4大支柱的部署模型，加强各个团队、组织和孤岛之间搭建桥梁。

# 利用有效实现DevOps的 4大支柱搭建桥梁

devops要保持有力而且影响长远，关键要素之一是它的灵活性。正如我们在这本书中展示的，没有一种实现devops的特定方法；它不需要某个特定的软件或流程，也不限于Web创业公司。

确实有一些故事 (Netflix和Etsy) 可以作为成功实现devops的范例。但是对于组织可以采用哪些方法使用有效实现devops的4个支柱提高他们的生产力，这些故事并不是无所不包的。有些组织 (如Etsy) 因其文化和技术实践而闻名，肯定有一些可以分享的重要故事，我们有意扩大了故事的范围，而不仅限于devops圈子里常常谈论的那些故事。这些故事的多样性并不影响devops的重要性，而是理解devops对于我们的工作方式以及整个行业重要性的关键。

人们在讲关于devops的故事时，通常会谈到孤岛。一个常见说法是开发和运维团队彼此是孤岛，相互割据，他们几乎从不沟通，更不要说有效地协作，人们希望找出办法来消除这些孤岛。我们更喜欢从一种更有建设性的角度考虑devops，而不是用一种破坏性的说法。

并不是看作为孤岛，我们只是把不同的团队或组织看作是岛屿。为了维护一个健康、生机勃勃的生态系统，多个岛屿需要能够分享资源，传递知识，甚至有居民在岛屿之间移动。因此，我们需要在这些岛屿之间搭建桥梁。建的桥越多，我们的岛屿网络就越健壮。利用有效实现devops的4大支柱所做的事情就是如何在不同个人、团队和组织



# 故事的意义

在这本书中，我们强调了故事的重要性和意义，既然已经熟悉了我们的4大支柱，你可能想看到这4个支柱如何共同影响你自己的故事，无论是过去还是现在。

人们讲的故事就是容器，承载着他们的世界，为他们的生活赋予了意义。

——Andrew Ramer

从某些方面来讲，devops就是要理解以及可能改变我们关于同一性的核心理念。如果我们基于角色形成自己关于同一性的理念，并因为一些人的行为与我们的理念不一致而将其拒绝，这会影响到我们赞赏哪些工程师、认为哪些候选人更适合我们的职位、如何面试以及接受哪些人。采用devops时，不是说“我是运维，因为我本来就做这些事情”，我们应该说“我是运维，因为我现在做这些事情”。这会把我们引向我们鼓励的成长型思维模式，而不是一种僵固型思维模式：不是说你“做还是不做devops”，这实际上只是关于如何分析和解决问题。

要了解为什么devops在业界这么受关注，我们可以从多个角度分析它的意义。作为团队和组织的中心，它有一种功能，会影响个人以及整个行业的生活和工作结构。研究不同的公司文化使我们有机会与各种各样的人交流和沟通，它还使我们有机会了解devops与将出现的更新理念之间的关系。

作为个人，在分析、倾听和分享从个人经验得来的故事时，我们会增强对社区的归属感，适应共同的团队价值观，并对出现的事件得到额外的认识。作为一个团体，我们通过共同的文字改善沟通，基于共同的理解减少冲突，并通过共同的价值观和理念增加凝聚力。

## 显性和隐性故事

尽管人们总是喜欢通过讲故事的方式来沟通，但这并不是展示我们的历史和文化唯一的途径。

显性故事（Explicit stories）以一种直接叙事的方式讲故事。这是最常见的，而且通常会作为例子反复讲。我们会有意地讲这些故事。

隐性故事（Implicit stories）会分享关于我们的文化、历史和行为的有关信息。这些故事并不直接讲。

在讨论我们的devops旅程时，我们通常不考虑隐性故事。例如：

## 鼓励应聘者加入我们的团队

在面试过程中，我们通常会不知不觉地传递公司价值观的有关信息。如果提到周末工作时间很长，这可能会发出一个信号，意味着工作和生活可能不平衡。这也说明人们对这个产品热情很高。

## 写博客文章

文章中包括哪些信息以及写到什么程度，这会向人们传达关于知识的侧重和期望。

## 在行业大会上展示

人们在行业大会上展示的水平 and 角色可以体现出信任和透明度，另外如果鼓励全公司的个人贡献者发言，还说明没有声望阶梯。

### Alice Goldfuss, New Relic网站可靠性工程师 (SRE)

对我来说，devops就是把开发和运维综合在一起，来创建可靠的软件 and 平台。这包括自动化和测试以及妥善的事故管理。

不过devops可能远不止这些，它可以是整个文化。毕竟，团队在解决共同的问题之前需要先相互理解。实际上，devops文化就是这样一个有效的途径，开始时我甚至没有意识到我已经在做这方面的工作。当然你要与其他团队沟通，事故应当是无问责的，而且当然需要多样化的人才，这些都是理所当然的。

没有夸夸其谈，也没有正式的启动仪式，但我所在的公司将devops价值观结合到日常的工作当中，我很幸运在这样一个公司里。我们会完成无问责事件回顾，在开发团队中加入SRE，并尽可能提供透明度。例如，我们的工程组织由一个流程仓库控制，任何工程师都可以对此做出贡献，我们会发布每月时讯，提供最新的流程变化。

我感觉可以与公司里几乎所有工程师交流，人们总是愿意倾听并乐于帮助我完成当前的项目。由于每个生产工程团队都轮班待命，或多或少地，我们会讲相同的语言并彼此提供支持。实际上，我们在朝着一个界限不明确的技能集合发展，软件工程师可以排查解决Linux问题，而SRE可以编写系统工具和Web应用。

这就很完美吗？不，当然不是。如果你凌晨3点被短信吵醒，让你解决别人代码中的问题，你肯定火冒三丈。不过，如果有避免这种情况的正式流程，说明没有人会因为临时情绪而受到伤害。如果一种文化支持人们对不同学科的兴趣，意味

着人们会留任更长时间。另外，如果认识到平衡的团队的价值，这会给初级人员留下空间。

如果这就是devops，那么每个人都应该实现devops。

## 理论和实践中的devops

讨论一件事理论上怎么做是一回事，放在实践中则是另一回事。如果一个人对某个软件做了某个修改，可能会对自己（或者队友）说“理论上这应该没问题”，但是对于这个修改在生产环境中的表现还是会有一点不确定。

关于如何看待这个世界以及事物如何运转，我们都有自己的思维模式。不论是否意识到（实际上我们通常都意识不到），这些思维模式会引导我们日常生活中的思想和行为。这称为使用理论，或使用我们的理论或思维模式时它们如何表现。不过，在问及如何看待这个世界或者我们认为给定情况下要如何表现时，我们通常会给出不同的答案，也就是我们如何考虑我们的行为或者希望有怎样的行为，这称为信奉理论，这与我们真正怎样做往往不一样。

大多数情况下，人们的信奉理论与使用理论不相同时并不是刻意想要欺骗。这是人类的天性，认为我们应该采用更好的、更积极的、更理想的方式，而不是我们面对具体情况的压力和现实时所采取的实际方式。所以尽管一个管理者可能声称处理一个故障时他当然会让下属确定最佳的处理方法，但实际上可能会陷入微管理行为模式，事无巨细都亲自处理，而且他甚至注意不到这种差别。

## 真实案例：展示实践的故事

这本书中已经分享了一些真实的案例，展示devops实践使用理论的现实性和变化性。声称你有一个无问责环境是一回事，真正有这样一个环境则是另一回事。



参加会议研讨或者阅读其他组织如何实现devops的博客文章并与你自己的组织做比较时，一定要记住这一点。看起来其他公司的文化遥不可及，很容易让你产生挫败感，但是要知道他们的信奉理论和使用理论很可能是有差别的。

你可能看到这些人或组织在宣扬他们的工作，或者有人称他们是“行业的领跑者”，这也会让人很困惑。你可能会想，“嗯，这对他们很有效，不过现实是这些在我的组织中并不可行”。由于一些特定的挑战，有些组织中实现改变可能更为困难。



尽管我们认为可以完成更多的文化改变，比人们所能想到的还要多，但可以肯定的是，不同的组织和行业会有不同的需求和限制，并不是你想完成的每一个改变都可  
行。如果必须保持PCI合规性，例如，肯定有些方面需要工作流的改变，如完成信用  
卡处理的服务器，而对此你确实无法改变，另外还有一些限制必须遵循。不过，这并  
不是说大多数组织在别的方面没有显著收益。

## 从故事学习

整体论是指一个整体中的部分紧密关联，因此不能视为它们的简单累加。这也适用于有效实现devops的4个支柱：尽管它们可以单独考虑，不过devops运动的强大之处在于对所有这4个技术及其交互的理解。

通过故事，我们可以学习：

- 为什么选择某些特定的工具或技术。
- 人们相互之间如何交互，以及如何使用不同的工具来达成他们的目标。
- 在真实世界里工具如何促成（或阻碍）目标实现。
- 不同的团队和组织如何共同解决不同的问题。
- 哪些方面有效，更重要的是，哪些方面不起作用以及为什么。



下一章中，我们将讨论团队或组织中存在的不同形式的知识和学习，还会介绍可以完成哪些改变来鼓励更多的亲密性和学习。

## 用故事建立关联

除了考虑一个组织中不同部分如何相互关联，还要考虑我们，作为从业人员以及作为人相互之间如何关联，这也很重要。如果只考虑软件或技术，而完全不考虑创建软件以及使用软件的人，这是不可能的。我们的软件是人创造的，而且要提供给人使用，如果将讨论的重点完全放在技术方面而忽略人的方面，最终看来是短视的。

故事可以帮助我们建立相互之间的关联。在第1章中分享我们自己的故事时，就是希望读者能够与我们建立关联，包括理解我们是从哪里开始的，并开始形成他们自己的故事，与我们齐头并进。分享故事会让我们把对方看作是真实的人，而不是一个匿名

的标识或者只是一个名字和一个卡通形象，通过分享故事，我们会开始交互，相互之间建立同理心。

通过分享和讨论这些不同故事中发现的丰富的经验，我们希望你能：

- 意识到其他人不同的经验会导致文化的差别。
- 学习如何问问题来确定一个声称使用devops的组织是否言行一致。
- 提高对其他人观点的容忍度。
- 研究其他人和自己的经验，比较和对比多种观点。
- 增强能力来形成自己的理念和价值观。

在这本书最后两章中，我们会分析各种影响我们个人的故事，这些故事对我们如何共同工作产生的影响，以及这对于建立可以让人们蓬勃发展的健康、可持续的组织有什么意义。

## 小结

你可能会奇怪这本书为什么会讨论这么多文化因素，而没有花更多时间来考虑技术因素。文化定义为一个组织中人们共同的价值观、理念、目标和实践，相比于特定的工具或技术，文化对于devops的实现会有更大的影响。

正如我们之前讨论devops组合概念时所说的，devops是为了建立理解和共同的目标，使个人和团队之间有长期、可持续的工作关系。如果一个人对基本工程概念有很深入的理解，可能很容易学习和使用一个新的编程语言，类似地，如果组织能深入理解人们共同工作的文化方面，就能更容易地调整使用不同的工具和技术。

为了建立有效DevOps需要的关系和关联，我们必须能够互相学习，并相互之间建立关联。故事可以为我们提供这种学习和关联的机制和途径，不论这是Usenet上的故事、早在2009年第一届devopsdays大会上分享的故事，还是这本书里讲述的故事。

# 搭建DevOps文化桥梁： 从故事学习

故事是学习中很重要的一部分，不论是对讲故事的人还是对听故事的人来说都是如此。你可能认为学习只是要了解如何使用一个新工具、利用一个新的编程语言，或者改进某种技能，不过与技术细节一样，在一个特定环境中如何使用以及为什么使用不同工具和技术的上下文也会带来影响，甚至影响更大。

幸运的是，故事可以作为一个很好的途径，可以用来分享在一个特定环境中使用工具的文化上下文。例如，Netflix的Chaos Monkey通过一个让应用随机地破坏虚拟服务器来测试生产环境中的故障。Netflix分享了使用Chaos工具的故事，这些故事体现了这个组织的价值观，具体包括：

- 让工程师在一天中精力最充沛的时候和凌晨2点时解决故障。
- 编写降级而不是失败软件的标准。
- 预期故障也是一种软件运维模式。

在这一章中，我们将介绍这个文化上下文中展示团队或组织价值观的不同方面，包括隐性和显性地展示。然后，我们会介绍在团队之间甚至组织之间鼓励学习的一些方法，讨论如何在你自己的环境中促进这种学习。

## 什么故事可以让我们了解文化

正如第1章我们指出的，文化中很大的一部分包括人们共同的价值观、规范和知识。



谈论文化是一回事，不过，具体看到或听到这个文化在日常工作中的体现又是另一回事。

这一节会分析文化的5个关键方面：价值观、禁忌、故事、仪式和想法。我们会讨论如何在日常的工作环境中建立这些方面，并对如何在你自己的文化中检查这些方面给出建议。关于我们的文化，这些正是我们希望其他人了解的方面，不论他们是刚加入组织的新人，还是你在大会上发言时的听众。



要记住重要的一点，你可以在所讲的故事中使用这些文化方面，不仅如此，在你自己听故事和从故事中学习时也应当注意这些方面。你听别人（正式或非正式地）讲一个故事时，他们谈到哪些关于文化的内容？他们的文化中哪些部分是隐性而不是显性体现的？通过注意文化上下文中学习哪些要素最有价值，你能更好地从他人的故事中学习，也能更好地讲你自己的故事让他们学习。

## 价值观

每个组织都有价值观，不过理论上描述的价值观与实际中体现的价值观并不总是一致的。价值观是原则、行为标准以及一个组织中哪些重要而哪些不重要的判断。

组织的价值观如何在组织内部以及在组织之外传达很重要。最常见的，会把信奉价值观写在某个地方。在类似公司网站或其他宣传材料、招聘启示、员工手册或激励海报中可以找到这些价值宣言，其中往往有类似“客户满意度”或“团队工作”之类的说法，也可能在全组织大会或新闻发布会之类的场合中口头重申这些价值观。

### 理论和实际价值观

这些口头和书面的描述指明了原则，但是在一个组织的日常生活中，理论与实际分歧最大的方面就是所表现的行为。

很多人都听过这样的说法，“你视而不见的标准就是你接受的标准”。

在工作场所中也是一样。设立和履行行为标准的责任尤其要落在那些领导角色或大人物身上。这很重要，因为这可以避免归咎于那些遭受不良行为的受害者（遗憾的是这种情况相当常见），而应当由那些有权力有义务履行行为和设立标准的人负责。

不过，这并不是说只有当管理者或领导人看到不好的行为发生时才可以发声。如第三部分中指出的，团队作为一个整体所履行的行为和结果对于维护全体有益的行为标准最为有效。这说明，有能力这样做的每一个人都应当尽自己的努力来设立和履行价值

观。这也能避免受到不良行为的人（这通常是边缘化或弱势群体中的成员，他们对于做什么没有多少权力和安全感）独自承担其他人恶劣行为的责任。

体现价值观的另一个方面是整个组织中如何对待不同的团队。同样在第三部分中指出过，在创业公司里，工程团队（尤其是Web和移动开发团队）通常会比其他非工程团队更受重视。尽管这通常不会明确指出，但是通过一些行为可以明显地反映出这一点，如工程师可以有更灵活的计划安排或更多远程工作机会；为他们提供更高的出差、培训和旅行预算；另外对他们的成果给予更多的认可。

## 区分团队和组织价值观

从整个组织的价值观转向更细粒度的团队层次时，各个团队之间可能有截然不同的价值观，这也可能导致整个组织出现冲突。我们在前面已经讨论过，不同的价值观是促使devops运动发展的核心。快速交付特性和网站稳定性就是不同工程价值观的例子。不过，工作如何完成或者如何评价并不是团队和组织唯一需要考虑的问题。还有其他一些价值观的例子，包括：

- “快速行动，打破陈规”。也就是说，相比所有其他方面，更看重向前推进。
- 重视教学与分享。个人/集体知识。
- 重视构建一个包容而多样的团队/快速发展一个团队。
- 鼓励人们畅所欲言/创建一个让人们感觉安全的环境。
- 强调作为团队成员/作为“单一个体”。
- 强调工作质量重于工作的时间长短。
- 鼓励人们在办公室吃一日三餐/鼓励他们与家人共度时光。

团队之间这种价值观的差异可能会导致冲突，特别是在不同团队必须紧密合作时。关键不是从一开始不允许有任何差异或不一致，而是团队要如何沟通以及出现不一致时如何加以解决。

人们是否愿意或者能够与其他人沟通个人和团队价值观，这是一个很好的指标，可以反映出这个组织能否让一群不同的人有效地合作，而且他们往往有自己不同的工作方式和价值观。这又回到我们在第一部分中介绍的devops组合的概念，开始时的沟通用来建立共同的理解，这不仅是对目标的共同理解，也是对将一同使用的一般策略的共同理解，信任的环境则允许人们或团队朝着这个目标较独立地工作。

如果对目标、策略或价值观没有共同的理解，显然这是不可能做到的。如果人们从来没有讨论或沟通过，如何知道在朝着哪个方向努力？一个组织原先越割据，沟通越少，他们就需要更多的沟通来建立共同的理解，而且要花更多的时间来达到一个可以保持这种状态的信任、无问责的环境。

## 沟通和传递价值观

在团队、个人和组织之间如何传递价值观？如何确定我们的价值观和目标有多少重叠和共同点，如何最好地传达对我们来说什么最重要，以及如何解决不同价值观之间的冲突？

### 让人们做好准备

需要做这种讨论时，要告诉人们为这些讨论做好准备。让关键的利益相关人定义他们的价值观是什么，解释为什么这些很重要并确定其优先级。如果只是说“因为原先就是这样的”，这在试图处理价值观的不一致时并不能帮助解决冲突。要确保涉及的每一个人都有机会分享和讨论他们的价值观和观点，保证每个人的声音都能被听到。

### 提高沟通技能

这可能表示管理层培训，使整个组织的管理者在有效处理冲突方面具备相应的技能，或者是为所有层次的个人贡献者提供沟通培训，但是重要的是，要从一开始就让人们培养好的沟通习惯，并尽早消除不良的习惯。

### 尽可能面对面交谈

不论是真正的面对面交谈还是通过视频会议，重要的是，要记住我们的沟通很大一部分是非口头的。书面沟通会丢失很多上下文和细节信息，所以更容易产生误解。对于更习惯书面沟通的人，可以利用准备步骤写出他们的想法，之后再当面表达他们的观点。

### 文档、审查和迭代

人们不会一直记得某个会议或谈话中讨论了什么，计划也可能意味着某个人无法真正出席。另外，理解错误也是有可能的，对于所说的和所达成的一致意见，人们可能有不同的解释。通过书面记录所发生的事情，就能进行审查，并为继续交流提供一个可靠的起点，从而继续达成共同的理解。

### 保持信息可见并分享

最后一点，要让每个人都能看到所进行的交流以及达成了什么决定，这很重要，尤其是有关于影响其日常工作的价值观。要想真正地分享价值观，就不能有隐



藏，或者关起门来做决定，然后直接作为通告下发。如果没有可见性，就不可能有交流，而交流对于共同的理解是必不可少的。



一般地，完全可以使用你在技术工作中所用的相同的工具和一般策略来完成这种构建社区的“软”技能工作。人们已经熟悉这些工具的一般 workflow，不仅如此，还能减少学习使用一个新工具时可能带来的磨擦。例如，如果你的团队已经在使用GitHub的拉取请求来协作完成代码工作，他们也可以使用同样的方法协作完成明确团队或组织价值观的文档。

在这个行业中，我们很高兴地看到已经逐步开始讨论可持续的工作实践，不再推崇长时间的工作，而是开始分享处理策略并鼓励我们的同事休假。假装不存在压力、工作过度、倦怠和其他类似的问题并不能让这些问题真正消失，这只会让人们在苦苦挣扎时害怕提出或寻求帮助。考虑你的团队和组织价值观时，要确保不要忘记人的方面。

## 禁忌

禁忌是已知或已经描述为危险或要禁止的事物。不过，关于这是隐性知识还是显性知识，在不同的团队中可能有很大区别，因为它通常以部落文化的形式出现，而没有明确的说明文档。环境中的一些禁忌例子包括：

- 在生产环境中作为root 运行命令而不是使用sudo。
- 在生产环境中测试对生产配置的更改（即使这只是一个监控脚本）。
- 未经测试就将代码提交到版本控制。
- 直接部署而没有经过测试。
- 运行来自互联网或公司系统的任意代码。
- 在周五或即将下班回家之前部署到生产环境。

这些禁忌可能是技术方面的，也可能属于非技术方面。重要的是要记住，禁忌越明确，就越容易向加入团队的新人传达这些禁忌。如果没有写下来或者明确地说明，人们就会在事后才知道自己有过失。

在这个领域，不同背景的人可能有不同的社会或文化期望，这会带来影响，如询问（ask）和猜测（guess）文化（在第二部分中介绍过）。明确阐明和描述关于价值观和禁忌的期望非常有助于促使devops组合自我强化。

技术性禁忌可能会在代码注释中指出，或者更常见的，可能会在wiki页面或其他共享文档中说明。描述如何在生产环境中部署代码的wiki页面可能包含类似“要当心X”或“警告：继续之前一定要确保Y发生”等禁忌。这种禁忌通常源于之前犯过的错误，所以记录下来希望能避免这些错误再次发生。

非技术性禁忌通常采用员工手册或行为守则的形式。尽管可能还有其他一些书更深入地解释为什么行为守则很重要，不过这里我们要说明的是，行为守则对于详细描述失当行为、违规执行政策、报告违规行为的流程以及给定环境中违规行为的后果是必不可少的。所有公司都应当有一个员工手册，而且所有活动都应当有行为守则，明确描述哪些类型的行为是禁止的，并给出示例、违规的后果以及如果看到这一类行为如何报告。

## 描述和说明禁忌

对于技术和非技术禁忌，在描述时要尽可能具体，这会很有帮助。对于更人文方面的禁忌尤其如此；如果只是说“不要做无赖”（这种情况很多），对于一个人怎样才算是“无赖”会有很多种解释，而且这会让报告一个违规行为变成一个人对另一个人的个人攻击，在给定的社区或组织中，由于这个原因，没有太多权力或特权的人通常会认为报告违规不太安全。另一方面，如果禁忌明确指出“在交谈或演示中不允许出现某些方面的内容”，这就清晰得多，可以让人们清楚地了解什么是不允许的。

很多情况下，解释为什么存在一个特定禁忌可能会有帮助。这不仅提供了上下文，可以指导人们决策，还会让规则看起来不那么随意。如果人们了解了规则背后的原因，通常就不太会违反这些规则和禁忌。对于技术禁忌，可以采用例子的形式，即如果做了一件禁止的事情会发生什么（或者提供一个链接，可以回溯到导致这种做法成为禁忌的那个事件）。对于行为守则之类比较社会化/非技术性的禁忌，一些人可能认为不需要指出为什么这些禁忌是必要的，但我们认为强调每个人的健康和安全性再多都不为过。

最后，要注意如何具体执行禁忌和规则，这很重要。你的环境是一个无问责环境还是会找某个人当典型？结果或反应与原先声明的是否一致？例如，如果行为守则指出违反这个行为守则的人会被开除，确实这样做了吗？如果不遵守这种禁忌会发出一种信号，认为这些禁忌并不重要，所以如果你不打算真正执行，就不要设置这样的禁忌。



另外，要注意执行规则时是否不一致。如果一个组织中看起来允许某些人违反规则，这不仅会为组织里的其他人树立一个不好的榜样，而且还会建立一个不好的环境，不同的人有不同的标准，这会让所有人失去安全感。



## 故事

在这本书中我们已经多次提到故事的重要性，也谈到它们对文化的影响。故事是在一个文化或社区中分享的经典故事或理念，它会解释“为什么”并影响行为，但通常并不以事实数据为基础。

### 故事的有害影响

故事是否有害要依情况而定。比较轻松的故事可能有些调侃的意味。有些故事可能会导致更长期的问题，涉及人们如何思考，相互之间如何交互，以及如何处理相关的其他行业等等。

作为有害的说法，一个经典的例子就是“女孩子数学差”。这种现象在第9章介绍过，被称为成见威胁。它描述了这样一种现象：如果被告知或提醒存在负面成见，在这样的环境中人们可能会有更糟糕的表现。这很大程度上是因为故事对我们产生的心理作用，可能是负面的，也可能是正面的。人们要考虑证明或反驳其他人对他们（以及他们团队中的其他成员或与他们类似的人）的负面认知时，会有巨大的压力和心理负担。

另一个有害的说法是“我不是技术人员”。在一个不断给工程师加光环的行业中，工程学科之外的人可能不再相信他自己技能的重要性，还会低估他们在工作中的贡献。工程师不应被捧得高高在上，而忽视其他员工，因为要发展和维持一个成功的企业，只靠工程技能是不够的。

这个说法的另一个问题是它指示了一种僵固型思维模式而不是成长型思维模式。人们往往会说“我不是一个工程师”或者“我不是技术人员”，就好像这是一种无法改变的事实，这会打消他们的积极性，而甚至不会去学习新的编程或运维技能。如果认为工程师比任何其他人都更重要，这种说法又会进一步导致工程师不愿意学习他们所在组织中的业务或客户相关方面的任何内容。与“开发”和“运维”相比，这些技术和非技术孤岛更宽泛，不过它们仍是孤岛，可能会妨碍一个组织充分发挥其最大潜能。

### 分析故事

我们遇到类似这样的故事时，应当问问自己，不仅要问这些说法可能带来什么危害，还要知道如何才能克服它们的影响。对于行业之外但是想进入工程领域的人来说，有各种编码训练营项目，这对于他们提升技能很有帮助，不过我们还需要考虑如何对待我们行业和组织中的非技术人员。如何鼓励和发展非工程师的技术和工程能力？如何提升工程人员的业务素养？在面试过程中要怎样做来克服成见威胁的负面影响？



不同的组织和公司都会有他们各自的故事，不过重要的是，要关注这些故事，检查它们对行业中的不同团体和社区所带来的影响，并不断质疑和迭代调整以改进我们所讲的故事。

## 仪式

仪式即一个团体或社区的成员定期参加的正式行为模式，这不仅对建立社区很有用，对确定一个社区的价值观也很有帮助。从社会学的角度来讲，正式仪式的想法通常有一种特定的含义，不过行为不一定要那么正式才能认为是一个仪式。如果能够在社区成员之间很好地分享知识，而且会定期开展活动，这样一个活动或行为就可以认为是仪式。

很长时间以来，仪式都用作将社区凝聚在一起的一种方式，通过参加仪式可以帮助建立一种共有的身份。这方面一个很明显的例子就是学校的“新生入学活动”，通常会有一些恶作剧。参加这些共同的传统活动，一旦仪式完成，就会建立一种友情和团体归属感。

### 社区内的仪式

我们在前面几章讨论过，建立社区和培养团体归属感是组织中发展强协作关系的关键，为了做到这一点，仪式可能是一种很有效的方法。不过，重要的是要知道我们创建的仪式是否排外，会把哪些人排除在外，以及这种排外对个人和整个社区可能有多大的危害。下面来考虑一些例子，很多技术公司里都可以看到这样一些仪式：

#### 经常工作到晚上9、10点

有些员工除了工作之外还有其他的责任义务（如需要照顾家庭），对于这些员工来说，类似这样的仪式是有问题的，不仅如此，这还会促成一种工作与非工作之间缺乏平衡或边界的文化。如果管理者也一直工作到很晚，即使他们口头上说这并非必要，但人们肯定会认为这种行为是必要的。

#### 喝酒或在酒吧庆祝达到里程碑或目标

庆祝是认可人们完成工作的一个很好的途径，不过庆祝有很多不同的方法。以酒为中心的文化可能只是在酒吧庆祝，或者鼓励过度消费，这不仅对不喝酒（或喝得少）的人不友好，而且是不健康的。

#### 在聊天中使用表情包和流行文化

随着chatops越来越流行，分享表情包、流行文化热点或其他内部笑话会变得越来越常见。同样的，尽管公共笑话可能是一种在团队成员和团队之间增进友情和

加强纽带的方法，但是如果有人不能理解这些引用或笑话，就会感觉被排除在外，另外这也会混淆工作中哪些可以接受而哪些不能接受的界限。对一个人看来是无害的笑话或GIF，对另外一个人可能会是侮辱性的。为了与大公司的严格管制有所区别，很多创业公司可能会采用一种“天马行空”的态度，不过这并不利于建立一个真正包容的工作环境。

### 在办公室举行100个俯卧撑比赛

这种活动也有助于加强团队纽带，不过不一定能让各种不同体能水平的人完全参与。尤其在员工平均年龄比较年轻的创业公司，团队活动可能更倾向于某个特定子集喜欢的活动。诸如梦幻运动队；足球、乒乓球或桌球；以及健身都可能营造一种“技术男”的氛围。并不是说不允许这些活动，毕竟要找到每个人都喜欢的活动通常是不可能的，不过重要的是，要注意活动和仪式的类型和种类，以及这些活动是否会无意地倾向或排除哪些人。

### 工作中提供免费餐和零食

免费餐可能是一个很好的福利，但提供早餐和晚餐可能会给人一种印象，似乎鼓励员工留在办公室的时间更长，而不是鼓励一种工作与生活的平衡以及有工作之外的爱好和兴趣。

### 指定时间可以带狗或孩子来

同样地，要记住，尽管有些人可能喜欢狗，但并不是每一个人都爱狗（而且有些人还会过敏），这很正常。这可能也是一种加强员工之间纽带的好办法，但是要有指定的无狗区或专门的育儿空间，尤其是在一个开放的办公环境中，否则噪音很容易会让员工分心。

讨论这些仪式时，我们的主题越来越明显，就是关于平衡和体谅。前面描述的所有仪式可以适当使用，由于有共同的经历，这会把人们凝聚起来，从而促进协作和亲密性，不过仪式的某些方面可能会让人感觉不舒服或被排除在外。通常这种排外是隐性的，可能是无意的，但是有时在“文化契合”的伪装下会显现出来，要避免这种情况。应当确保各种仪式尽可能包容。

另外一类仪式与组织如何工作有关：CEO可能每周固定时间在一个公开论坛上回答问题，VP可能约定每个季度与部门中的每个人共进一次午餐，或者可能大力鼓励员工参与支持工作轮班，从而对客户面对的问题有所认识。与前面描述的仪式一样，这种仪式有助于建立公司文化，不过这些仪式往往没有太大问题，因为它们涉及与工作更相关的问题而不是社会化问题。

## 改变和创建仪式

关于组织或文化仪式和实践，最后要考虑的一点是改变或创建仪式的频度，以及是否在整个组织中明确地传达了这些改变。如果一个公司的仪式五年未变，会发现这些仪式已经不适合，不再反映当前的组织价值观或人们对组织的期望。一个正在成长的组织应当鼓励创建新的仪式，要纳入新人和新的流程，尤其是在多样性和包容性方面。

要不断检查我们的仪式，了解它们对工作生产力以及员工如何交互的影响，另外要了解这些仪式为什么重要和有意义（或者不重要）。“总是这么做”并不能作为以某种方式处理一个技术问题的可靠理由，而且事实上，对于人文方面的仪式来说，这也不是一个好理由。

## 想法和知识

“总是这么做”并不是继续这么做的一个好理由，而且这也说明存在一种僵固型思维模式，而不是成长型思维模式。经常出现这种情况的一个领域就是“最佳实践”。最佳实践是通常被接受或认为是最正确或最有效的想法或过程，但是这种思维在Web运维或现代软件开发等飞速发展的领域中可能是有问题的。

人们寻找最佳实践想法有很多原因。它们可能是最小化风险的方法，而采用其他方法时（如增加监控或部署更小的变更集）可能存在这样一些风险。如果有人要进入一个之前没有任何经验的领域（对于有很多小团队的创业公司来说，这很常见），他们往往会寻求行业最佳实践作为指导。这也是人的天性，总是更愿意相信一些实际用过的最佳方法。

### 寻找“最佳”想法

这里的问题在于，通常并没有做某件事的“最佳”方法，也不存在一种“最佳”解决方案可以同样地适用于每一种情况。如今随着我们的产品和架构越来越复杂和多样，有更多不同的可变部分，另外有大量不同的技术可以从中选择，对一个组织最适用的方法可能对另一个组织完全无效，而且某个阶段最适用的实践做法在6个月或12个月之后可能就不再是最佳的解决方案了。

人们希望解决认知失调，对于不相容的想法就会产生这种心理冲突。如果我们可以从“最佳”文化转换为一种“当前适合”或“设计模式”文化，就能缓解将来必须把一个“最佳”方法替换为另一个也是“最佳”的方法时出现的失调。如果某些做法没有达到我们的预期，或者随着需求和限制的改变，这些做法的有效性也会改变，这就会



带来认知失调，倘若我们不再使用“最佳”作为一种不可变的、绝对的标签，就能避免这种失调。

是什么让人们接受一个想法是“最佳实践”或一个“当前适合”的解决方案？对于如何接受证据，人们可能表现出不同的偏好，就像他们的工作和学习方式也有自己的个人偏好一样。对有些人来说，一个权威的数字就足够了。有些人则需要亲自实践的直接经验才能接受这种方法确实可行而且能很好地工作。这些需求可能会带来冲突，这取决于我们如何沟通。



对于更倾向于直接经验的人来说，要让他亲自尝试每一种可能的解决方案是不现实的，但是对他们这种偏好的了解会影响其他人的沟通方式，如会包含已经做过哪些尝试的更多细节，提供额外或背景阅读材料的链接，或者增加图表或其他度量结果。

## 思维模式和学习新想法

人们对于如何、何时以及为什么寻求新知识和新想法可能有很多不同。毫不奇怪，在这方面有僵固型思维模式还是成长型思维模式会有很大影响。僵固型思维模式往往不鼓励人们寻求或接受新知识。如果一个有僵固型思维模式的人认为自己很聪明，并形成这种自我意识，要学习新东西或改变他们对某个事物的想法就会对这种自我意识提出挑战；有僵固型思维模式的人往往会更强烈地拒绝新想法，特别是那些与之前知识相抵触的新内容。

另一方面，有成长型思维模式的人可能把他们的成功看作是一种学习和努力的结果，而不是某种天份，这些人通常不仅更愿意寻求新知识，而且也更愿意接受这些新知识。有成长型思维模式的人对所有健康组织的继续成长和提升都是必不可少的。幸运的是，完全可以把僵固型思维模式转换为一种成长型思维模式，有关这个主题的更多内容参见第20章。

## 组织间的交互

除了来自一个组织内部的故事和经验，在组织之间分享故事也很有意义。组织如何实现这一点可以作为一个很好的指标，从中可以看出这个组织是否能成功地建立持续的devops文化。

可以看到，就像个人一样，组织也可能有僵固型思维模式或成长型思维模式。一个有僵固型思维模式的组织会把他们的成功看作是固有的、必然的，不论是多年来一直很

成功的企业，还是刚获得风投资金的新初创公司，都有可能存在这种僵固型思维模式。这种僵固型思维模式可能导致他们忽略一种信号：他们在抗拒和阻挠对现在做法的所有改变。

一个组织如果有成长型思维模式，这个组织往往更关注继续学习和提升，认为成功需要不断的努力，而无法得到保证。这些组织会寻求新想法，尝试新的解决方案，并在技术和文化方面寻找更好的方法来完成工作，而不是认定目前的做法就是最好的。

组织如何寻求新信息？他们与其他组织有哪些交互？要进行这些交互和信息交换，最常见的方法通常是通过行业会议、较小的社区活动以及工程交换项目。

## 会议和旅行

参加会议是人们走出自己的组织，向业界其他从业人员学习的一种很有意义的方法。人们可以参加关注某个特定技术的会议，如一个特定的数据库解决方案或者编程语言，或者参加涵盖更宽泛主题的活动，如移动开发、Web性能或Web运维。会议发言可能包罗万象，从技术性主题到文化性主题都有涉及，而且这些会议的走廊会很有意义，即常常在走廊上（或者餐桌旁或喝咖啡时）进行的临时性会谈。

### 旅行的成本

不过，去参加会议是有成本的，这不只是机票、酒店、误餐补贴和地面交通以及参会本身的费用（如果你的组织目前没有涵盖这种参会费用的培训或教育预算，现在就应该开始做这样的预算）。参会旅行还有一种心理、情感甚至身体上的成本，对于参会者和发言者都应当考虑这一点。

对于所有需要旅行才能参加的会议，人们必须对孩子的照顾、宠物的照顾或房屋打理做出安排，这些开销通常不会涵盖在组织的旅行政策中，这可能会对承担主要家庭责任的人（通常是女性）带来更大的负担。经常旅行可能因为旅行的舟车劳顿和离家在外带来焦虑或压力。与朋友、家人和同事分开可能导致孤立甚至关系紧张，特别是如果一个人频繁的旅行，家中其他人就要比从前更多地照顾家庭和孩子。时差和疾病也是需要考虑的问题。

很多组织都有旅行的相关政策。通常政策包括每个员工每年有固定的预算参加培训和会议，每个团队或每个部门有一个类似的预算，另外每个员工可以参加一定次数的活动，或者有某种规则对不同的人按不同方式承担旅行开销。例如，“你可以参加X会议（由会议承担旅行和住宿费用），作为发言者可以参加Y会议（由公司承担费用），作为参会者可以参加Z会议”。

创建或检查旅行和培训政策时，要记住，并不是每个人都有能力让活动方支付他们的机票和酒店费用。很多会议根本不会支付这些费用，只有人们提前协商好才有可能支付（这对有些人有不利影响，即使协商也效果甚微，而且协商往往会带来不良后果）。或者只会负担比较热门、更有成就的发言人的费用。如果是后一种情况，这可能意味着你的组织会反复派出同一个人，而不是给经验较少的发言人或更初级的员工同样的机会参加会议。

### 会议安全的考虑

如果希望人们参加会议时要在会议上发言，尽管这可能有助于减少组织承担的费用，因为发言人可以协商会议报销旅行和住宿费，但这会让不想发言的人无法得到很好的学习和交流机会。并不是每一个人都希望公开发言，尽管这种能力肯定是可以学习的，但是有些人更愿意以其他方式回馈社区，如写博文、写技术文档，或者对开源软件做贡献。

作为行业中的弱势或边缘群体的成员，公开发言或作为发言人还有可能受到不好的对待。一个组织不能只是为了让人参加一个会议或培训而使他们置身于感觉不安全的环境。

遗憾的是，作为一个弱势团体中的成员，即使只是单纯的参加会议也可能是一种不太好的体验。如果你的组织有一种政策，要求公司或团队中只有一个人可以参加某个活动，可以考虑一下，如果他们与认识和信任的某个人一起旅行，他们可能会感觉更安全（而且确实也更安全）。如果员工在旅行和参加行业活动时遇到安全问题，要确保他们有必要的求助资源（现场或远程），而且允许他们采取临时措施从而能更安全地旅行。

最后，要记住，如果一个人参加会议的目的是去发言，与去招聘或者只是去学习相比，会议会有不同的体验。发言人会发现，由于紧张，在他们发言之前，他们自己可能不太能集中精力听前面的演讲，而如果你派出员工参加会议的唯一目的是招聘未来的新员工（或者销售某个产品），那么重点也不是他们学习的能力，尤其是如果这意味着只是布一个展位而不是参加演讲。



会议是在行业中分享知识的一种很好的途径，而且应当作为组织或团队预算和计划的一部分。要确保允许人们参加会议（强度要适当），至少每年参加一次活动，使他们能提高他们的技能，扩大他们的社交网络，并提升团队和组织知识。



## 其他社区活动

其他较小的社区活动（如线下聚会小组）也是在行业不同组织之间分享知识的好方法。由于线下聚会会有一个本地中心，通常不会有相关的旅行和住宿费用。尽管也可能出于其他原因旅行的同时参加一个线下聚会，但大多数情况下人们只是参加他们当地的聚会。

这种小组通常比一般的会议规模小得多，不过对于参会者和主办方来说，都能大大减少开支。很多情况下，会有一个当地组织允许聚会小组在他们的办公场地举办活动而不收取任何费用，只是要求留出几分钟讨论公司提供的就业或销售机会作为交换。举办或者甚至只是参加这些活动是一种很好的招聘方法，可以从中发现未来的新员工。

大多数较大的城市里，已经有很多关于各种主题的小组或活动，与更大的会议相比，这些活动种类更多，也更专业，这是因为创建一个小组或活动的开销要低得多。由于开销更低，这也使得启动新的小组更容易，所以如果你居住在一个技术背景较少的地方或者要找一个特定的技术但还没有相应的小组，完全可以启动这样一个小组，这是为当地社区提供和分享知识的一个好办法。

邀请一系列发言人可以帮助你了解另外的观点和分享知识。可以只让发言人在组织内部的会议中演讲，但是这并不能与整个社区的其他部分分享——而且如果没有一种互惠或回馈意识，一段时间后，可能没有人愿意去这样一个组织发言。要在公开活动中邀请发言人（而且可以在线提供视频，包括现场或事后提供），如Etsy的Code as Craft系列发言人，这不仅对你的组织有好处，对其他组织也有好处。

与举办这种较小的活动相比，运营一个公开的技术博客开销甚至更少，这允许组织与社区的其他部分分享知识，而且你会发现有些员工可能更愿意写而不是在公众面前讲。如果希望在行业内提升组织和相关文化的知名度，这样一个技术博客可能很有用（这对于招聘也是一个加分项），它可以为新员工提供背景信息，并鼓励其他组织开始分享他们的故事。

## 工程交换

我们在第三部分中讨论过，工程交换项目就是两个不同组织的工程师在某个很短的时间内相互交换职位和角色，如果执行得当，这会是组织之间分享想法和知识的一种非常好的方法。

与会议讨论或博客文章相比，通过交换项目可以对组织如何运作有更细致的了解，获得更切实的认识。在会议讨论和博客文章中，可能有一种趋势，希望向大众呈现好的

一面，往往会剔除不太完美的细节，并不是有意隐藏，但是可能不会完全分享。这么做并不是错误，但是对于相互学习来说，如果能看到和了解出问题和不太完美的部分可能会得到更全面的认识。

## 工程文化和开放性

再次提出，一个组织对交换项目的开放程度会决定这个项目是否成功。如果一个组织不允许外来的工程师做任何实质性的事情，或者学不到什么，将来的交换项目就不太可能再邀请这个组织参加，因为他们没有履行他们的义务。这种交换也是一种合约或社会契约；必须有共同的理解，双方都必须参与，而且可能在内部对不公平的行为会有惩处。

是否允许工程交换项目，或者是鼓励还是阻挠，这会很好地体现组织的思维模式。要注意一个交换项目双方如何对待参与项目的工程师：

- 给定组织中是否有些团队或管理者更愿意参与？
- 是否仍希望参加交换的人在交换期间在线、查邮件，并做他们平常的工作，或者是否允许他们更充分地投入交换项目？
- 交换的工程师是否会因其参与工作得到正面评价，或者是否会被这个组织看作是不友好的人？
- 通过交换项目得到的新想法或建议是否直接（或极快地）被忽略，还是会给予充分地考虑？
- 允许交换来的工程师完成多少工作？是有实际意义的工作还是只是繁琐的杂务？
- 是否允许交换来的工程师在会议中发言，还是只允许他们听会？
- 会留给这些工程师一些东西（贴纸、马克杯或T恤衫之类的纪念品；他们记的笔记；他们完成的一部分重要工作），还是只让他们空手而归？

如果你的组织或团队感觉停滞不前，或者反复争论而无法达成任何共识，纳入一个有全新视角和观点的新人通常会带来一缕清风，这很有必要。尽管甚至可以在交换项目中引入一个具有一组特定技能的人，但更值得推荐的做法是不要利用交换项目来解决特定的问题，而应当尝试一种成长型思维模式，并探索新的可能性和想法。

## 鼓励组织间的亲密性

如果你的组织有一种僵固型思维模式而不是成长型思维模式，你要怎么做？如何在开始分辨你的组织是否有僵固型思维模式？

### 阻止僵固型思维模式

僵固型思维模式最常见的症状之一是刻板地坚持一种“我们总是这么做”的想法。这并不是说只能使用最新最前沿的技术；实际上，有时要坚持使用经过反复尝试和验证的技术，而不是在生产环境中试用时间还不到几个月的新技术，这是很有意义的。如果一个组织希望基于实际解决问题的情况来评价和评估技术和工具，而且会根据很多问题做出决定，那么坚持使用他们已经在用和了解的工具和技术会更好，这是可以的。但是绝对拒绝任何改变，甚至未经仔细考虑就选择拒绝，尤其是过程的改变，这就说明存在僵固型思维模式。

另外要注意诸如下面的态度：

- “这对于Facebook、Netflix和Etsy很适用，但我们可不是那种公司。”
- “我们本身就是高绩效的组织；我们不需要devops。”
- “这是一个企业，那种事情对我们不适用。”

要知道你的组织的优点和缺点，你希望维护怎样的工程或组织价值观，以及过去哪些对你不适用，尽管这很重要，不过还要关注一些说法，这些说法可能暗示了一直以来都这么做，而且将来也会这么做，而不只是目前的做法，这一点也很重要。把它与一个人的僵固型思维模式做个比较：“我数学不好”与“我现在正在努力学好数学”。从运维层次上讲，这可能就像是“devops对我们不适用”与“我们正在想办法让人们更多地协作”。

### 从小的改变做起

要从一种僵固型思维模式转换到成长型思维模式，最值得推荐的一种方法是使用小的重复动作，另外通过较小的频繁的成功来强化新习惯和思维模式。在组织层次上，这可能包括：

不要试图一蹴而就地完成改变

很多组织的僵固型思维模式可能来自某些人，他们可能对之前的改变有不好的体验，或者出于各种原因拒绝风险。从小的改变开始会有帮助，不要一次彻底改变



整个部署系统，可以先从一部分开始，即使这可能只是修正不正确的文档。寻找可以改变和提升的小问题，这会帮助建立信心，相信改变不仅可以带来改进，而且不一定意味着大规模出现问题或故障。

### 强调过程，而不是最终结果

如果一个组织的管理层或领导认为devops只适用于创业公司，而与他们无关，试图推进“devops变革”、建立“devops团队”或者达到类似的目标就不太可能会成功。在这种情况下，把重点放在你想要改变的行为上，这可能是实现配置管理、改变人们的监控方式或者解决最大的痛点问题，并强调这些改变的原因，而不是过于急切地描述“宏大蓝图”。

### 只从一个团队开始

改变需要成本，一个改变的影响或作用范围越大，成本就越大，这不仅包括实现这个改变本身的成本，还包括如果从某种程度上讲这个改变是一个错误而带来的风险（正是因为这个原因，持续交付较小的代码更改往往是一个好主意）。不要试图一次改变整个部门或组织，先改变一个团队作为开始，这样更易于降低成本和风险。你可以尝试先从一个比较独立的团队开始，尽可能减少对组织中其他部分的影响，不过用一个与其他团队有交互的团队来试验也有好处，这样其他人就能看到和感受到改变带来的好处，可能开始对采用新工具或实践变得有兴趣。

### 养成学习的习惯

要改变思维模式，不论是个人的思维模式还是组织的思维模式，都涉及改变我们的习惯，而成长型思维模式的关键习惯是学习。要鼓励人们在每日站会或每周状态会上分享他们学到的有意思的事情，即使是学习某种并不可行的做法。建立一个邮件列表，人们可以在这里分享有趣的文章或他们读到的博文，还能与其他人讨论这些文章。鼓励人们参加当地聚会并分享他们从这些会议得到的收获。建立一种学习的好习惯会逐步显现出它的好处，并有力地推动更大规模的组织学习和文化改变。

总的来说，要克服组织间建立亲密性的阻力，最好的方法之一就是了解导致这种阻力的思维模式。管理者可以建立会议预算或者建立一个发言人系列，这会有帮助，不仅如此，个人贡献者也可以展示在不同公司和组织之间分享知识的好处，这可以最小化风险，减少恐惧，还能帮助文化改变真正发生。

## 小结

正如我们在这一章中展示的，文化可以有多种表现方式，包括价值观、禁忌、仪式、故事和想法，因此，一个组织也可以采用多种方式开始建立devops文化。有一些需要人们和团队谨记于心的常见主题，其中最重要的是个人和组织的思维模式组合会影响（形成或破坏）文化的成长和学习能力。

僵固型思维模式（也就是人们匆匆地下结论认为“这对我们肯定不合适”和“不过我们总是这么做的”）不太可能做出显著和可持续的改变，这种僵固型观点会成为一种自我满足的预言。另一方面，如果有一种成长型思维模式，其重点在于个人和组织学习，而且要通过个人和集体努力来实现成长和改变，他们要做的改变更有可能成功。

即使是成长型思维模式，要实现一种协作和亲密性的文化并不断为此做出改变，这也不是一蹴而就的，这些改变对于不同的组织可能也不同。要记住这一点，而且要记住这是很正常的。并不是每一个组织都必须有相同的价值观或相同的文化，不过明确你的价值观对于建立你想要的文化很关键。一个组织肯定会有自己的价值观，不论它是隐性还是显性的，不过价值观越明确越清晰，学习起来就越容易。

“这是一个企业，那种事情对我们不适用。”

许多组织在开始建立devops文化时，会遇到各种障碍。这些障碍可能来自组织内部，也可能来自外部环境。例如，组织可能缺乏必要的资源，或者组织可能缺乏必要的领导。此外，组织可能面临来自竞争对手的压力，或者组织可能面临来自客户的要求。所有这些障碍都可能阻碍组织建立devops文化。

建立devops文化是一个长期的过程，需要组织持续不断地努力。

组织可以采取多种措施来建立devops文化。例如，组织可以建立devops团队，或者组织可以建立devops社区。此外，组织还可以采取其他措施，如建立devops文化大使，或者建立devops文化奖励机制。所有这些措施都可以帮助组织建立devops文化。

不要试图一蹴而就地完成改变。

许多组织的僵固型思维模式可能来自其领导人。他们可能对之前的改变有不好的经验，或者出于多种原因拒绝风险。最小的改变开始会有帮助，不要一次彻底改变。

# 搭建DevOps文化桥梁： 发展人际联系

故事除了帮助我们学习成功的技术和有效的文化，还允许我们相互之间建立和维持强联系。在第三部分我们讨论过，个人和团队间联系的强度会对一个组织的健康和生产力整体产生积极影响。

故事允许我们在个人层次上培养这种联系。要了解我们自己的个人故事中看重什么，个人故事就是我们所讲的自己的故事，这会帮助我们更好地了解和理解其他人。在这一章中，我们会讨论这样一些个人故事要素，以及它们与建立devops的文化环境有什么关系。我们还会介绍这些个人故事积累起来对组织健康会有什么影响，以及如何创建健康的文化体系而避免不健康的文化。

## 关于工作的个人故事

有些人会回避在工作场所与人交往过密，他们会说“我只关心工作”，或者更看重技术技能而非其他方面。不过，除非你是一个单人组织，只为你自己创建软件，否则肯定会与其他人一起共事，而且要为其他人创建软件。工作中的人际交往方面是绝对不容忽视的。

在这一章中，我们将分析工作中不同形式的个人故事，以及它们对组织文化产生的影响，或者组织文化对这些个人故事的影响。从人们加入一个组织到他们最终离开这个组织，这些个人故事是组织文化的一个关键部分，而且对于在这个组织中工作的人来说，这些个人故事对组织文化是否健康也有重要影响。



## 个人故事的价值

考虑到个人的长处和观点，并允许对工作最熟悉的人推荐改进方法，这一直是成功 devops 环境的基础之一。实际上，这个运动最早就是由注意到过程中存在缺陷的工人发起（具体来讲，这个缺陷就是存在孤岛，导致开发人员和运维人员之间缺乏协作和沟通），并开始考虑如何改变他们的工作方式来改进这些过程。

### Hollie Kay: DevOps

DevOps是过去几年里技术剧变带来的最具变革性的事物之一，对我来说，它最好的一点是可以打通派系隔膜，所谓派系是一种割裂的文化，孤立的开发部门和系统管理部门之间就会出现这种文化。我一直在努力将不同类型的好技术汇集在一起，缩小它们之间的差距。

对于devops概念来说，是工具更重要还是文化更重要？几乎从devops术语出现以来这个问题就一直存在。我们已经在这本书中做了解释，基于文化对人们如何工作、为什么工作、工作时如何交互及如何做出工作相关决策（包括使用哪些工具和技术以及如何使用）的影响，文化更能体现devops运动的精髓。

文化显然与人紧密相关。同样的工具和政策如果由不同的人群使用（他们有不同的目标、工作方式和理解），可能会得到完全不同的文化或工作环境。我们讨论的工作是由人完成的，而且最终要用于其他人，若非如此，根本不会有这一类谈话。因此，在第二部分已经讨论过，一定要考虑人们如何工作、他们是怎样想的以及是什么促使他们这么想，这很重要。

文化还与价值观紧密相关，这包括个人层次和组织层次的价值观。要认识到个人的重要性，尊重他们在自己的工作领域的经验和知识，允许人们建立自己的个人故事并影响其团队和组织的故事，对于一种健康有效的devops文化，这些都是很关键的价值观。在接下来的小节中，我们将介绍这些价值观如何隐性和显性地体现。

## 表彰个人

哪些人最容易得到表彰？从这一点可以很大程度地反映出一个团队或公司的价值观。表彰新聘用的人员时就能体现这一点。通常，表彰最多的新聘人员是那些著名人物或业内知名的“明星”，他们可能是公司花大价钱从其他公司挖来的。这就会发出一个信号（尽管通常是无意地）：这些著名人物比其他人更重要，而且把他们“挖”来比保留现有的员工更有意义。如果一个新聘用的“明星”会得到丰厚的奖金和薪水，而

这些是从原本为现有员工提供的预算中拿出的，本来要用于加薪、发奖金以及有利于现有员工的其他方面，这种情况下，不满和人员流失会迅速增加。

更高级的角色通常也会得到更多的表彰，随着职位的提升，这种有意或无意的偏向性会导致公司变得越来越单一，这意味着更小范围的人会得到更引人注目的表彰。尽管没有人希望因为他们所属的团体而受到注意或得到表彰（如果有人是因为人们认为“太好啦，我们终于聘用了一个女性”而受到关注，就好像这是她唯一值得一提的特点，她就能证实这一点），但还是应该表彰不太醒目或比较初级的人员，这很重要，这样可以表现出他们在组织中也受到重视和欢迎。

人们加入一个组织时，就是在加入一个成熟的社区。我们如何欢迎新人加入社区，这可以很好地体现出这些社区的健康程度。我们与Nicole Johnson对她关于devops及其社区的看法进行了交流，她是Chef的一位企业领域解决方案架构师，过去8年一直致力于基础设施虚拟化、云计算、运维和自动化等不同领域的工作。

### Nicole Johnson, 企业领域解决方案架构师

认识到devops的巨大威力很重要，不只是对于有经验的从业人员和社区成员，我们还欢迎新的devops社区成员加入，而不论他们的背景或行业是什么。我们看到大型企业（传统行业，如银行和制造业）可以从devops受益。这并不是让一切都自动化，而是要采用一种适用于整个组织的方法，可以让每个成员都作为团队中有价值的重要部分。要认识到，有效的协作不一定能利用工具实现，关键是要采用devops文化，打破组织中的孤岛，并最终充分利用机会实现协作。

我从一个比较传统的技术领域转向Chef的一个采用devops的组织，在这个过程中，对我来说有一点很清楚：全力以赴地采用这种工作方式才能让适当的环境完成有意义、可持续的转变。我曾经在很多组织工作过，他们的主要绊脚石不是技术上的问题，而是组织问题，主要在于他们的割据方法。克服这些障碍是推进IT和组织转变至关重要的步骤。

多年来我一直都非常关注系统部署、应用部署和测试，从我的经验来看，正是devops帮助很多组织完成了自动化的最后一步。他们未达到预期的原因通常不是因为技术上的绊脚石，而是因为孤立的操作和缺乏协作。

作为devops社区的一个新成员，我清楚地看到，devops是允许组织成功转变其运作方式的要素之一，为此要采用一种协作文化，另外要结合使用基础设施作为代



码、自动化和持续交付。之后我很快了解到，对于devops还有很多工作要做，我很高兴这个社区一直都如此热情。

一个人对于社区的第一次体验（不论这是一个都是新同事的组织、之前从未参加过的会议还是世界各地devops从业人员的社区），会对他关于这个社区的看法以及是否认为自己是其中一员产生持久的影响。



作为团队现有的成员，遇到新成员并与他们交流时要记住这一点，这很重要。我们可能会说自己很看重多样化的观点，但当我们与其他人想法不一致时，可能并不一定能尊重他们的观点，这也是隐性价值观的一个例子。

## 晋升

另一个很容易体现价值观的领域是组织内部的晋升。宣布有哪些晋升以及在多大范围内宣布，由此一种文化也能隐性地体现其价值观，特别是如果这个过程中存在着不公正的情况。如果完全由管理者决定如何宣布，也会出现偏向性（不论是否是有意），可能导致这样一种情况：男性的成功会比女性的成功得到更多关注，或者某个团队比组织中的其他团队得到更多认可。这种情况会让一些个人或团队感觉他们的贡献不被重视而滋生不满。

感觉得到认可是预测工作满意度的关键因素之一，所以要采取一些措施，如关于如何宣布晋升制定一个政策，这会使人们感觉得到了公正的对待。这个政策可能只是一个简单的原则，如“在某个人所属团队的下一次周例会上宣布他晋升到X级，而在下一次部门范围的会议上宣布他晋升到Y级别”。

要对整个组织如何处理职业发展制定标准和政策，这会让历史以来都不平衡的竞争环境变得公平，还有助于在组织中培养一种公平和欣赏的文化。

## 采用远程方式

随着类似旧金山和纽约等较大技术区的生活成本不断攀升，越来越多的人希望生活在不那么拥挤、更舒适的地方。不论是希望在郊区生活的夫妻，还是只是厌倦了将大把收入都花在房租上的单身，可能都有很多合理的理由希望能远程工作。原先由于技术上的原因，本来可以远程工作的一些职位无法真正做到这一点，但是由于高速互联网连接几乎无处不在，视频会议软件也在不断发展，这样的情况越来越少（有些角色可能需要做诸如测试大量硬件的工作，如果硬件太过庞大，运送不太现实，或者如果必



须在一个实验室里测试，这可能不适合远程工作，另外数据中心的技术人员必须在数据中心附近，但是大多数以软件为中心的角色都完全可以远程工作）。

刚接触远程工作的管理者经常会提出这样的反对，“如果我不能走到他们的办公桌旁边看他们在工作，我怎么知道他们确实在工作呢？”想看到人们的工作，即工作的可见性是一个有意思的想法，但是实际上，管理者无法区分一周工作80小时的人和只是假装工作80小时的人之间的差别。



在办公室里假装工作与远程假装工作同样容易，所以如果你想了解完成多少工作，可以考虑采用其他方式回答这些问题，如在面试过程中了解，而不是要求每个人都在办公室里工作（而且办公室通常成本很高）。

尽管远程工作在如何支持协作、沟通和工作的可见性方面确实存在其特有的问题，但如果一个组织拒绝员工远程工作，甚至对那些完全可以远程工作的职位也是如此，这就会在很大程度上体现这个组织的价值观。通常这是一个隐性而非显性的价值判断。这种文化通常会坚持原有的过程，“因为总是这么做的”，即使这些过程完全可以改进。不允许远程工作反映了这种文化更看重工作的形式而不是实际工作的有效性。在办公室待多少个小时或写了多少行代码之类的指标并不能很好地说明工作质量。

如果有人想要改变团队或工作场所，从组织如何回应这个人就能很好地说明这个组织的健康程度和灵活性。如果一个公司使用不适当的指标度量人们的工作，如强调每个人要在办公室待多少个小时，而不是他们的工作质量或者是否在最后期限之前完成工作，很可能会导致实际工作输出减少和员工满意度降低。我们在第二部分讨论过，在一个团队或项目中，如果让多种工作方式协作，这有很多好处。对于什么时间工作以及在哪里工作，如果要求每一个人都一样，而不是允许采用不同的工作方式并从中学习，就会抹杀不同工作方式的这些差异。

最后，要考虑如何在整个组织传达变化，如人员在团队间转移。在这方面，公司支持多大的透明度或者是否允许透明，这些可以很好地体现这个公司实际的透明度（尽管公司可能在理论上声称很重视这个方面）。是否允许个人公开谈论他们希望换团队或改换工作场所？他们身边的同事是否知道可能发生这些改变，或者他们在这些改变发生之前是否一无所知？谁把这些改变传达给部门或组织的其他人，是改换工作场所的那些人、他们的管理者，还是根本没有人传达这些变化？

## 离开公司

有人要离开公司，这也是一个需要沟通的领域，如何传达这种改变可能会对周围的人 and 团队有很大的影响。首先要注意的一个问题是希望人们给予多大的关注，不只是HR或他们的经理，还包括与他们工作最紧密的那些人。取决于个人的职责范围，要离开的人可能有大量工作和知识需要转交给其他人。同样的，这也能很好地体现一个组织看重什么：是注重“作为团队一员”的形式，还是更关注透明地分享知识，使团队成员能够更有效的工作？

一些组织极力阻止人们分享他们要离开的消息，因为他们认为这个消息对于团队中其他人的士气会有不好的影响，不过我们坚决反对这种做法。人们会离开公司：这是一个简单的事实，有过工作经验的任何人都能理解。不让人们谈论他们要离开，这会大大限制工作和知识的有效转交，使其他人没有机会询问所转交工作的有关问题，而且像这样保守秘密对士气可能有更多的负面影响。有人要离开，把这样一个简单的事实当作一个秘密来保守，这不利于建立一种信任的文化，人们可能还会开始怀疑是不是还有其他信息对他们隐瞒。

同样的，关于人们的选择以及这些选择背后的原因允许有多大的透明度，这与组织整体的透明度是一致的。如果有更大的透明度，这会使组织中有更多的信任，反之亦然。对于如何处理人员的离开，如果有人是自愿离开，而有人是被开除或要求他离开，可能会有一些区别，但是沟通的程度和方法可以反映一种信任和诚实（或者缺乏信任和诚实）的文化。

### 为什么人们离开组织

可以跟踪人们为什么离开，这会是一个很有价值的信息来源，可以从了解一个组织在哪些方面可能存在问题，前提是人们能够而且愿意诚实地讲出他们的原因。遗憾的是，一个组织了解这些理由并最能从中受益的情况往往正是人们最不愿意给出理由的情况。如果有人感觉窘迫或者不安全，而且对于这种情况没有任何解决办法，他们会认为在离职谈话时解释这种情况可能不安全。或者，有人已经反复尝试改进有问题的过程或者力图修正文化问题，但是没有任何进展，他们可能认为再解释这些问题没有任何意义，也不会有任何机会做出改变。

不过，有一些常见的离职理由值得考虑，因为这些理由反映了组织整体的文化，而不是单独的个人原因：

## 不尊重人们的时间

不尊重员工在办公室的工作时间就已经够糟糕了，不过如果希望他们把办公室之外的时间也用来工作，对于那些有其他兴趣爱好或责任的人，肯定就会开始考虑离开。如果人们有其他的兴趣爱好，能适当休息、放空自己和充电，他们通常在工作时会更专心，更有生产力。另外，如果一种文化偏向那些工作之外没有任何其他责任的人，这往往很快会成为一个由年轻单身异性恋男性作为主体的单一文化。我们可能会说，公司一开始就不应该希望人们在工作时间之外加班或者查email（极端场合和待命轮班时除外），但是如果确实有这样的需求，就应当在招聘启示中明确地指出，并提前给出理由，这样人们至少能够准确地了解他们要做的工作。如果没有做到这一点，就会导致倦怠以及人员流失。

## 不尊重别人

如果人们没有感受到来自同伴、管理者或组织的尊重，倘若他们有其他选择，就不太可能在这个组织工作太长时间。通常人们并不是要离开他们的工作，他们是为了离开现在的管理者，更常见的情况是，当人们感觉到管理者不尊重或不信任他们时，他们就会离开，而不是由于意见不一致或者有个人冲突。如果聘用某个人来完成某个工作，但是被管得太细，甚至导致工作无法正常开展，他们就会感觉自己的技能在别的地方更能得到认可。如果感觉管理者没有为他们辩护，或者没有以某种方式支持他们的职业发展，也会出现同样的情况。

## 没有获得信任作为回报

任何关系中的信任都必须是双向的。组织（尤其是创业公司）通常对员工要求很多，比如工作时间比原先商定的更长，参与他们不感兴趣的项目（只是因为这个项目对企业很重要），甚至在财政紧张时还会减薪。通常这里存在个人风险，尤其是那些作为边缘化团体成员的员工，这种风险要求人们必须获得信任。如果人们认为公司不会给予这种信任，或者认为领导层没有考虑到这一点，他们通常就会去其他能够得到信任的地方。

即使人们没有明确地指出这些理由，你也可以找寻一些规律，如某个经理手下多个人离职或人们频繁离职、某种文化使得很多人都要晚上和周末加班，尽管并没有“官方”的明确要求（要记住隐性期望会有一种显著的效果，特别对于没有太多经验的员工或者有些人可能认为需要更多地证明自己），或者离职发生在组织领导或方向发生重大改变期间或之后。这可能是一个信号，说明社区与其成员没有达成应有的一致。



# 文化负债

技术负债是指技术决策的最终结果，如系统设计、软件架构、软件开发或技术选择，而文化负债用来描述文化决策的最终结果，包括聘用和解雇决策、创建和执行哪些社区标准、组织层次架构和价值观。

技术负债的有关想法也同样适用于文化负债，具体来讲就是这些负债需要在某个时间偿还，如果没有解决相关问题，时间越长，累积的利息就越多，将来偿清债务就越困难。

文化负债的例子包括：

- 聘用一个大家都知道很难合作的工程师（或者这个人总是不断地捣乱，或者在公司活动中总是喝得烂醉），并要求其他人必须与他合作，否则只能辞职，而不是让这个人的离开或者改变他的问题行为。
- 存在太多的中间管理层，导致一些不必要的过程或者延长周期时间，而且根本不愿意重组或者减少人员。
- 一个会议或社区活动没有（或不执行）行为守则，所以大家都知道对于这个社区中弱势群体的成员，这会是一个不安全的地方，但是仍然以其他人的安全为代价允许有不良行为的人参加（或者甚至邀请他们发言），可能只是因为他们在社区里很有名。
- 邮件列表中允许不太友好的语言，使得新成员不太愿意加入，或者不愿意对这个列表做贡献。
- 组织中形成一种恶性循环，大家总是在工作，总是要在位，人们会在深夜发送 email 或做类似的事情，而且知道其他人也在工作，可能会快速做出响应，在这里没有人希望自己少工作而成为团队中的另类。



考虑你的组织是否健康时，如果希望改善组织的整体有效性，除了技术负债，考虑文化负债也很重要。

对于这一类情况，并没有任何行之有效的直接或短期解决办法。不要强制人们告诉你离职的真实原因，特别是如果涉及到信任或安全问题时。不过，你可以关注文化问题的其他指标，并继续培养一种安全、包容、无问责和信任的文化。

# 系统的健康

人们考虑一个组织或公司的健康时，首先想到的通常是财务健康。有多少利润和收入？年增长情况如何？市场份额或客户获取情况如何？有时组织或团队可能会考虑就业率或流失率之类的指标，不过通常认为这些不太重要，特别是财政情况很好时。

我们在本书前面讨论过，需要认真考虑诸如倦怠之类的问题，因为这些问题不仅影响团队和组织的士气和生产力，还会影响建立我们的组织和行业的那些人的健康。重要的是我们不仅要分析可能影响倦怠和健康的单个因素，还要分析系统因素。无论是着力让个人和组织福利提升的管理者，还是希望职业发展更健康、更平衡的个人，都需要能够明确一个系统或组织中的健康因素。

遗憾的是，这种信息可能很难得出，特别是如果你目前还不是这个组织的一员，就会更困难。人们通常不会讨论与工作相关的压力、倦怠、焦虑和其他健康问题，原因有很多：他们可能不希望被别人看轻，或者看起来另类，也可能不希望在一个专业场合讨论这么私人的问题（尽管在实际中专业与私人问题往往会有大量重叠），或者他们甚至不知道别人也遇到了他们同样的问题，这些问题很值得讨论。公司会在面试中或招聘网页上向未来的员工推荐自己，在这里不太可能提到高流失率、倦怠员工的人数或者文化中的其他“负面”问题。那么如何发现一个组织实际上是否健康呢？

## 检查病态系统

病态系统的4个性质或规则描述如下（这里是在一个工作环境上下文中描述）：

### 让人们太过繁忙而没有时间思考

如果工作场所中人们太过繁忙，而没有适当的时间来思考问题，他们就不太会意识到这个环境有多消极或不健康。这一点可以进一步扩展，包括使人们太过繁忙而没有时间交流，不鼓励在水房或咖啡间短暂休息，因为这是“浪费时间”，而且阻止人们与组织之外的人交流。最后这一点可能会以“公司忠诚度”作为理由，不过如果完全阻止人们对外交流，不论是技术问题还是社会问题的解决方案，这通常都不是一个好的信号。

### 让人们很疲惫

与让人们很繁忙类似，如果让人们很疲惫，也会使他们不会太多地考虑当前的情况。这不只是表示身体上的疲惫，还包括心理和情感上的疲乏。如果有人整天忙于救火，而且整晚上只有他一个人待命值班，那么不论在身体上还是情感上，他们都不会有精力来尝试增加自动化或修正一些不太好的过程，更不用说尝试完全

改变待命的处理方式或者去寻找另一个工作。我们在这本书中一直提到，要实现持久的改变，这需要不懈努力才能做到，而且不会一蹴而就，所以如果人们没有必要的精力来实现持久的改变和形成新习惯，一切可能都会保持原样。如果在一个工作环境中很疲惫，还意味着有人已经放弃或者接近倦怠，他们已经认定（不论正确与否）一切都不会很快改变，他们只是为这份工资而工作，而不是出于内在的动力。以这种状态工作的人越多，会进一步增加组织的情性，使整个组织更为疲惫。

### 让人们投入情感

人们投入的情感越多，他们就越有可能坚持某个方面。如果他们的自我价值与团队、产品或整个组织联系在一起，这一点尤其突出。在创业公司的早期阶段，这种情况相当常见，会有意地培养这种极端的忠诚度，出于对产品或公司的热爱，会促使人们放弃工作与生活的平衡、更有竞争力的薪水，真实的休假政策等。通过鼓励组织中的一致性以及个体与团体紧密相关，这会让人们有情感上的关联，而且会一直保持这种情感关联。为了实现这一点，组织可以为员工提供大量免费的定制品牌服装，使他们的衣柜里都是印着公司logo的衣服；通过提供早餐、正餐、干洗或其他福利，让员工在办公室工作的时间越来越长；或者在下班后“可选的”社交活动中做决策，使得那些不参加这些活动的人无从了解团队的最新情况。股票期权等延期的福利要在数年内才能行权，这会阻止人们离开这个组织，这也是让人们继续投入和投资的另一种方法。

### 间歇地奖励

间歇奖励的吸引力已经在心理学领域得到广泛研究，这方面有一个著名的例子，如果每次按杠杆时就会得到一个食物球，老鼠就只会在它们饿了的时候才按杠杆，如果按杠杆时只会间歇性地得到食物球，老鼠就会更经常地按杠杆，因为它们不知道下一次什么时候能吃到食物。尽管我们的想法无疑要比这些老鼠复杂得多，视频游戏和赌博就充分利用了这一点。在工作场所中，这可能涉及一些管理者可能采用不一致的做法，例如，对下属的反馈（如果有）可能很不一致，或者加薪、晋升和奖金系统完全不透明。

总的来看会怎么样呢？这可能是一个看起来危机四伏的组织，人们总是反应性地工作而不是主动工作，总是忙于救火而根本无暇顾及其他，无法未雨绸缪地在偿还技术负债方面取得任何显著的进步。这些救火工作可能要求人们一直要检查和回复手机收到的邮件、聊天消息和拉取请求，甚至包括晚上和周末，而不只是偶尔的短期加班。这样的一个组织中，员工总是太忙碌、太疲惫或者对公司过于投入，而很少或者根本没有任何爱好、兴趣或工作之外的活动。



对有些人来说，这样的一些例子听上去好像过于极端或者只是人为设计的，不过很遗憾，当前这些情况在技术行业的众多领域都太常见了。如果你没有见过或听说过文化负债或不健康系统的例子，甚至一个例子都没有听说过，这只能说明人们不愿意或不敢提出问题，要确保努力创建一个让人们感觉足够安全的环境，使他们能指出问题。

## 创建健康的系统

如果能发现这些特点（它们会使人们困在一个病态系统中，为一个没有发自内心真正考虑员工利益的组织工作），我们可以反过来考虑这些特点，明确如何让组织成为一个健康的系统。因此这些性质可以描述如下：

### 创造时间来思考和反思

不只是允许人们有时间思考，一个健康的系统还应当主动地鼓励人们思考。这可能意味着要求（或者强烈鼓励）管理者安排与所有下属定期一对一会谈，在办公室留出专门的空间让人们一起工作和交流，或者建立一个项目来搭配人员，可以根据正式的指导能力来搭配，也可以采用一种非正式的社交方式搭配，使人们能够分享不同的想法。人们应当能很自然地考虑问题的不同解决方案或者不同的合作方式。理想情况下，这种工作环境是无问责的，允许人们做试验，而不会因此受到责罚。管理者可以通过另外一些举措鼓励更多的反思和思考，如有专门的CEO办公时间，在这里组织中的任何人都可以与公司领导人交谈或问问题，从而建立双向的沟通而不只是一种从上到下的单向沟通。

### 鼓励人们休息和充电

要尽可能鼓励人们考虑他们实际上能做多少工作，而不要试图承担超量的工作。这种改变需要从上到下做表率，因为人们往往会以他们的管理者或领导人的行为作为典范，即使这些领导人说的与做的不同。晚上加班应该只是例外情况，而不能作为一般规律，如果这种情况开始经常发生，就应当主动修正你的计划过程，直到不再经常计划在正常工作时间以外安排工作。要努力偿还技术和文化负债，尽管这意味着要留出时间短期内先不做“常规”工作，目的是从长远来看创建更好的工具或过程。不要让人们疲于奔命，总是反应性地工作（因为他们根本没有时间主动地工作）。要主动寻求和倾听百忙中的人的解决方案，使他们有足够的权威、时间和资源来完成对他们有益的活动，这些活动不仅对他们的日常工作有益，对于支持组织的目标也会有帮助。

### 鼓励参与工作之外的活动

要鼓励人们休息好并主动地工作，而不是反应性地工作，与此类似，一个健康的

系统会鼓励人们培养和发展兴趣、爱好甚至工作之外的活动，要认识到这会让他们更快乐、更全面，而这进一步会使他们更有生产力。工作的质量肯定比在办公室工作多少个小时更重要。不要要求人们把他们全部的时间、精力或者特别是整个人都投入到你的组织或产品中，要有多种多样的经历和兴趣。要确保人们能很好地享有工作之外的时间，可以实现一个最小休假政策，同样要做表率，要让大家都看到管理者和领导会很好地利用工作之外的时间（而且完全抛开工作）。确保提供了足够的资源来帮助员工克服压力、焦虑或其他心理健康问题以及倦怠。

### 经常而公正地奖励

内在动力会比外在激励强大得多，尽管如此，在现实中，人们可能要付房租，可能要养家糊口，要对人们付出的时间和努力给予公正的报酬，这是他们应得的。确保你的组织中所有管理者都使用相同的过程、时间范围和原则来审查报酬，使不同管理者的下属在加薪和奖金方面不会有很大区别。组织应当确保他们有标准化的薪酬等级，薪酬制度使用要一致，而且要经常评估报酬，确保每个人都在适当的等级内。要记住，无意的偏见真实存在，女性通常不会协商，而且即使协商也会被区别对待。要确保你的招募人员和招聘经理不会使用这一点来“省钱”，对边缘化或弱势群体的成员给的工资低得多。不论你的加薪和奖金政策是什么，都应当明确而公开说明这些政策，以减少员工的压力，而不是让他们猜测。

如果有人要评估组织，比如有些人得到了多个工作机会，要在这些不同的工作机会之间做出选择，或者只是想确定是否还希望继续留任，就可以查看这4个性质，了解组织在健康和 unhealthy 这两个极端之间的哪个位置。一个组织完全落在某一个极端上是不太可能的，不过完全可以确定对你来说哪些方面最重要，并相应地优先考虑或评估。

## 组织和个人健康

在不健康的系统中，即使不像之前描述的病态系统那么不健康，也可以从其他方面看到对个人不健康的系统或组织因素。我们在检查工作场所的多样性和包容性时，发现了这样一个关键的例子。研究表明，对于在以男性为主的环境中工作的女性（当前几乎所有工程组织都是这样），她们几乎总是会表现出存在心理压力的信号（还要应对可能存在的任何其他形式的性别偏见）。

这种压力对健康会有显著而且持续的影响。在85%以上都是男性的环境中工作的女性（同样的，很遗憾，大多数工程部门都是如此）会持续显示出非正常的皮质醇曲线。皮质醇是一种压力荷尔蒙，随时间处于不同的水平，包括短期和长期，但是对于那些



“孤独”或“象征性”的女性，她们的皮质醇水平和曲线要远远高于那些处于常规日常压力模式的女性。一段时间后，这些女性的身体开始习惯于这种长期处于巨大压力之下的“工作或离开”生存模式，只有在离开这种工作场所转向一个更多样的环境很多年之后，她们的皮质醇水平才会调整和恢复正常。

如果皮质醇水平很高，这会对身体和心理健康产生非常负面的影响，导致或引发免疫系统变弱、甲状腺功能下降甚至骨骼、肌肉和结缔组织少量损坏等问题。尽管目前为止的研究只关注性别差异，但有理由相信，任何人如果处在这样一种位置上，感觉自己是一个“象征性的代表”，只有自己一个人或者类似的少数几个人来应对成见威胁，那么都同样会面对因为这种压力带来的影响健康的后果。

在一个组织中，如果缺乏工作与生活的平衡，其他与压力和健康有关的问题可能会因此更加放大。研究表明，压力大、“时刻在线”的环境往往会放大性别、年龄和其他形式偏见的效果。这一类组织往往最强调人们在办公室里工作多少小时，这往往会对人们产生负面的影响，最主要的可能是对女性和年龄较大的员工，他们在工作之外还有照顾家庭的责任。

在要求人们总是工作很长时间的环境中，往往喜欢那些不会质疑现状的人，如希望人们工作多长时间或整个环境的包容性。这对于长期的生产力和健康很不好，不仅如此，这种长时间工作的高压环境还会进一步增加一些人的压力，尽管这些工程师穿着帽衫看起来就像马克-扎克伯格，让人们认为他们也像他一样能干，但他们可能做不到这一点。



关于聘用“devops工程师”，一个常见的误区是你能聘用一个人，兼具这两个不同角色（开发和运维）的技能，他既可以是一个全职的开发人员，也可以是一个全职的运维工程师，不再需要为这两个角色分别聘用两个人，因此可以减少成本。这种理解是不正确的，也是不现实的，而且即使这是真的，这种情况也不可持续。试图让一个人做两个人的工作，从长期来看，这会导致低质量的工作，而且会导致倦怠，另外会让你的组织有一个对员工很差的坏名声。

这种问题的关键在于它们通常很微妙，一段时间后，这些问题的影响和强度会减弱，变得很难发现（不论是外部还是内部）。在病态系统中，“煤气灯光”经常让人们怀疑自己的认知和感受，导致他们认为是自己反应过度或者只是想象，而实际上并不是这样。在这种情况下，可能很难识别一个环境的负面性。另外，长期的压力以及对皮质醇和其他压力荷尔蒙的适应性会让人们感觉这样一个环境是正常的，但实际上这并不正常。



## 识别健康和 unhealthy 的文化

在加入一个组织之前，如何识别这是一个病态系统或不健康的工作环境？或者，如果这个环境随着时间越来越糟糕，如何注意到这一点并确定离开？除了前面描述的“健康系统”的特性，在你目前的环境中或者面试过程可能问的问题中，还要注意以下方面：

### 如何做出决策？

选择会带来影响，决策过程是否会根据所估计的这些影响而变化？或者是不是极小的选择也需要一个很费劲的审批过程？工程师（或任何个人贡献者）在他们的日常工作中有多大的权力做决策？比较初级的团队成员是否有机会通过试验做出自己的选择并处理结果，以此作为其学习过程的一部分？对于较大的决策，是否需要所有受影响的方面达成一个共识？有些人是不是有与其他人不同的过程？从一个组织中谁拥有决策权可以很好地看出这个组织的文化，以及在这个组织中做出改变是否容易。

### 一般的发布周期是怎样的？

除了比较直接的问题，如是否有自动化测试以及通常多久发布一个版本，发布周期还可以展示有关周期时间和交付时间的详细信息，这在第三部分中已经介绍过。正如老话所说，“不要让完美成为优秀的敌人”，这意味着试图达到“完美”或者总是希望“再多一点”，可能会妨碍你及时发布已经足够好的产品。一个团队或组织如果过度关注“完美”，很可能会发现自己迭代太慢，而不能获得或保持客户，尽管也可能有很多其他原因导致周期速度太慢。

### ticket 的生命周期是怎样的？

如果需要完成一个关于 ticket 的工作，谁来创建 ticket？如何分配 ticket：是创建 ticket 的人来选择分配的人，还是给定团队或项目的人们从某种接收队列中拉取 ticket，或者是不是每个项目里都有一个人审查和分配 ticket，还是综合上述方式？ticket 的优先级如何指定，另外对于不同的个人和项目，这些优先级是否一致？会多久为 ticket 分配一次到期日期？注意有些地方所有工作都有很紧急的到期日期，或者所有工作都没有到期日期，这可能说明在指定工作的优先级方面存在问题。

### 多大的风险算风险太大？

与如何做出决策类似，一个组织或团队如何响应风险也能反映很多信息。存在风险时，通常在哪些领域？这些风险是有关于使用一个新工具或技术，还是有关于客户是否希望或需要一个新的特性？是否相比之下有人有更大的自由可以冒险，

是否允许一个创业公司CTO大幅度重写产品，把这作为他的一个小项目的一部分，因为没有人能站出来对他提出质疑，而组织的其他团队却必须遵循必要的过程和规则才能做这种尝试？如果有任何类型的开发人员，他们不必遵守所有其他人要遵循的游戏规则，这对于团队或组织是有问题的。

在评估获得的工作机会或者考虑是否还留在某个组织中时，就要考虑系统是否健康或病态，这是一个很重要的方面。如果所有其他条件都相同，可以很容易地选择更健康的环境，不过各个组织很少在所有因素上都势均力敌，而且也很难从外部（或者甚至通过面试）了解一个文化的健康程度。如果你能坦率地与当前在一个组织中工作或者最近在这个组织工作过的人交流，这会为你提供你想要的有价值的见解。

## 小结

我们个人之间如何交互，另外如何与我们的组织交互，这是创建和维护组织文化的一个很重要的部分。我们所讲的个人故事（关于我们自己的工作）可以帮助我们识别其他人同样的故事，培养同理心并加强联系。

我们已经讨论了技术负债和文化负债的概念，它们分别是以往的决策对技术和文化的持续后果。这两种负债很容易产生，而一旦有这些负债，就很难消除。与之类似，坏习惯（如把所有情况都当作危机，或者整晚回复email而且每晚都如此）也很难打破。尽管每个人的故事对他们所在的组织会有显著的影响，但要注意，作为个人，你的第一要务应当是照顾你自己的健康，而不是你的组织的健康。

继续完成你的devops之旅时，要记住分享故事很有意义，这很重要。注意你的故事所体现的文化和相应的价值观，这包括显性和隐性的文化；联系和学习其他个人、团队和组织时是否有效；技术负债和文化负债会如何影响你的组织和人员的健康。



# 结论

这本书就快要读完了。祝贺你，另外也感谢你细心阅读这本书！我们在这本书中介绍了很多内容，分享了来自不同的人和组织的大量故事。你可能没有时间改变你想改变的每一个方面，或者并非我们介绍有效的DevOps文化时谈到的每一个方面对你的当前情况都重要。要记住，没有十全十美的解决方案。重要的是，你（你个人或者与别人一起）要区分出哪些是当前需要优先考虑的最紧要的问题，哪些是可以延迟到之后某个时间再做的改变，以及那些方面对你来说根本没有什么影响。

我们已经展示了，实现devops没有一条“唯一正确的道路”，没有devops成品或者devops作为服务。我们分享了一些想法和方法来改善个人协作、团队和组织亲密性以及整个组织中的工具使用，另外介绍了这些概念如何支持组织根据需要来改变、适应和调整。你已经了解，对于每一个希望提高其产品质量同时又改善员工效率和福利的组织，可以采用不同的方式应用这些共同主题。

不论你用什么语言编程，使用什么工具管理基础设施，或者是否使用最新最炫的容器技术，这些原则都是适用的。如果你能深入地理解有效devops的四大支柱如何协作，来最大化共同目标、促进共同理解并培养健康、可持续的工作价值观和实践，就能创建一种比任何工具或趋势更有生命力的文化。要充分理解人们如何共同工作以及如何维护和修复这些工作关系，正是这一点给予了devops不竭的动力。

devops并不只属于Web公司或小的创业公司，也不只适用于开发和运维团队。它不只是有关软件开发实践的一个重大改变，4大支柱中发现的原则和想法涉及一个组织的各个部分，甚至大型企业或政府部门也可以使用。这些理论可以采用多种方式付诸实践，重要的是不要过多地关注别人的故事而忽视了你自己的故事。



应该记得，不论你的组织的devops文化或旅程具体是怎样的，最终目标绝不是每天完成固定次数的部署、使用某个特定的开源工具，或者只是因为其他组织成功地做过某些事情就去做同样的事情。最终目标应当是创建和维护一个成功的组织，可以为客户解决问题。不论你在什么行业或者你的企业有多大的规模，都应当花时间主动地定义这些目标、价值观以及有助于实现目标的想法。不要等到发现隐性价值观已经形成，再想改变就太晚了。

## 接下来做什么

开始读这本书时，你可能希望得到一些指导和见解，从而能够在你自己的组织中完成有效的文化和技术改变。那么接下来做什么呢？

查看第20章中列出的材料。这一部分包含了我们认为很有用或者很有意思的大量图书、文章和视频、本书前面引用过的参考资料，以及我们认为很有帮助的一些材料的链接（对devops或其他文化改变感兴趣的任何人都能从这些材料有所收获）。

首先确定你认为你的组织中、你的团队中或者你自己的工作习惯中需要优先改进的最主要的问题。取决于你的当前状况，看起来可能要做太多的改变，很难确定从哪里开始着手，所以可以问问自己，你的日常工作中最让你沮丧的是什么，你认为你自己或你的同事在哪些方面不必要地花费了大量时间或精力，以及你认为哪些做法可以使你的工作立即变得更有效。

与你的团队和组织分享你的看法。找到共同的基础并达成一致，大家共同努力在组织中实现改变。不论你是希望做出重大改变的个人，还是公司的领导，都可以做出贡献而且确实会产生影响。

分析哪些改变可以单独完成，哪些要在团队的协作之下完成，另外哪些问题需要在组织层次上解决。在这个过程中要记住一点：你应当有适应性，能够开放地面对改变。人总是至少有一点抗拒改变，这是人类的天性，不过如果固守某些情况太长时间，以至于所有改变看起来只会变得更糟糕，这种趋势会恶化到丧失生产力的程度。

要注意已经掌握的习惯和思维过程可能会抑制有效的成长和改变。在确定需要优先考虑的紧要问题时，应当了解你在工作中的不开心或沮丧来自哪里，除了不合适的工具或过程，你可能还会发现习得性无助、习得性期望（可能是自己强加的，或者基于其他人对你的期望）或者习得性角色依从，所有这些都可能阻碍进一步的前进。要记住避免“我们总是这么做”的陷阱！

如果你在你的组织中有领导角色，那么还应当增加一项职责，要帮助组织坚定地建立适当的文化，使组织能稳步前进。要记住，devops组合依赖于共同理解。作为一个领导，要由你来帮助建立组织中的这种共同理解，定义成功地“实现devops”对你来说是什么意思。记住文化有价值观，不论这些价值观会显性还是隐性地体现，不过如果能明确地指出价值观，就能更容易地分析研究、讨论和改变。

尽管确实需要顶层的支持，不过要实现持久的改变，不能只采用一种从上到下的强制方式。影响整个组织的改变需要得到全面的认可，这意味着领导层需要帮助形成一个共同的声音，为此要寻找有效的方法使所有人的声音都能被听到，而不会阻碍做出决策。这也意味着要建立一个提供强有力支持而很少敌对的环境，在这里人们不怕问问题、尝试新事物或者谈论一些不可行的做法。

最后，要确保在需要时重读这本书中的有关章节，甚至要每六个月到一年将整本书重读一次，这样在你的团队和组织改变时，就会有全新的看法并重新评估进展。

## 创建有效的devops

devops不只是清单上的一项任务。当然你可以在你的组织中优先确定和完成特定的改变，但devops并不是一项能真正完成的任务。实际上，这是一个一直进行并不断迭代的过程。必须不断维护和更新共同理解，否则这种共同性不会保持太长时间。

另外devops也不是通过创建团队或给团队改名、把某些职位改成包含“devops”或者购买最新最酷的容器平台云服务就可以完成的。你无法购买或安装devops，因为它不需要（也不排除）任何特定的工具或技术。这也不是只与开发人员和运维团队有关，甚至不是只针对工程师。另外，不能通过把相应职责转移到其他人身上来实现devops。

实际上，devops有关于理解、同理心和相互联系，尽管可以一次优先考虑和关注它的不同方面或者这本书的不同章节，不过devops的强大之处实际上来自于所有这4个支柱的相互配合。综合在一起，这4个支柱可以形成并加强一种持续文化的基础，提供可持续的工作实践，并促进人们之间的关系。

devops鼓励组织中的每一个成员都做出贡献，来为整体提供价值。就像一个管弦乐队一样，要练习、沟通和协调，不会特别地打造几个“明星”。

devops涉及邀请参与持续的改变过程、认可组织中每个团队取得的成果，以及明确禁止不当的行为。就像一个花园，需要不断的施肥、浇水和除草，才能培育组织实现可持续的成长并获得商业成功。如果只是购买一束预定规格的花，不能把这看作是园

艺，类似地，如果只是购买一个声称是“devops解决方案”的工具，也不能认为这是devops。这是一个持续不断的工作，要建立和维护一种文化，使devops真正有效。

基于对本书中介绍的4大支柱的共同理解，我们可以改变我们的组织和行业本身，使它更有生产力、更可持续而且更有价值。请在我们的网站上([effectivedevops.net](http://effectivedevops.net))分享你的故事，让我们作为一个社区携手成长和学习。行动起来吧。



# 更多资源

## 什么是devops?

- Apache HTTP Server Project. “About The Apache HTTP Server Project—The Apache HTTP Server Project.” [https://httpd.apache.org/ABOUT\\_APACHE.html](https://httpd.apache.org/ABOUT_APACHE.html).
- ComputerHistory. “Jean Bartik and the ENIAC Women.” Posted November 10, 2010. <http://bit.ly/bartik-eniac>.
- Dekker, Sidney. *The Field Guide to Understanding Human Error*. Farnham, UK: Ashgate Publishing, 2006.
- Dekker, Sidney, and Erik Hollnagel. “Human Factors and Folk Models.” *Cognition, Technology & Work* 6, no. 2 (2004): 79–86.
- ENIAC Programmers Project. “ENIAC Programmers Project.” <http://eniacprogrammers.org>.
- Humble, Jez. “Continuous Delivery vs Continuous Deployment.” <http://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment>.
- Humble, Jez, and Farley, David. *Continuous Delivery*. Upper Saddle River, NJ: Addison-Wesley, 2010.
- Poppendieck, Mary, and Thomas David Poppendieck. *Implementing Lean Software Development*. Upper Saddle River, NJ: Addison-Wesley, 2007.

- Walls, Mandi. *Building a DevOps Culture*. Sebastopol, CA: O'Reilly Media, 2013.

## 协作：个人合作

- Friedman, Ron. "Schedule a 15-Minute Break Before You Burn Out." *Harvard Business Review*, August 4, 2014. <https://hbr.org/2014/08/schedule-a-15-minutebreak-before-you-burn-out>.
- Greaves, Karen, and Samantha Laing. *Collaboration Games from the Growing Agile Toolbox*. Victoria, BC: Leanpub/Growing Agile, 2014.
- Gulati, Ranjay, Franz Wohlgezogen, and Pavel Zhelyazkov. "The Two Facets of Collaboration: Cooperation and Coordination in Strategic Alliances." *The Academy of Management Annals* 6, no. 1 (2012): 531–583.
- Heffernan, Margaret. "Why It's Time to Forget the Pecking Order at Work." *TEDWomen* 2015, May 2015. <http://bit.ly/heffernan-pecking>.
- Hewlett, Sylvia Ann. "Sponsors Seen as Crucial for Women's Career Advancement." *New York Times*, April 13, 2013. <http://bit.ly/nyt-sponsorship>.
- O'Daniel, Michelle, and Alan H. Rosenstein. "Professional Communication and Team Collaboration." In *Patient Safety and Quality: An Evidence-Based Handbook for Nurses*, edited by Ronda G. Hughes. Rockville, MD: Agency for Healthcare Research and Quality, US Department of Health and Human Services, 2008. <http://bit.ly/comm-collab>.
- Popova, Maria. "Fixed vs. Growth: The Two Basic Mindsets That Shape Our Lives." *BrainPickings.com*, January 29, 2014. <http://bit.ly/fixed-vs-growth>.
- Preece, Jennifer. "Etiquette, Empathy and Trust in Communities of Practice: Stepping-Stones to Social Capital." *Journal of Computer Science* 10, no. 3 (2004).
- Schawbel, Dan. "Sylvia Ann Hewlett: Find a Sponsor Instead of a Mentor." *Forbes.com*, September 10, 2013. <http://bit.ly/hewlett-sponsor>.
- Silverman, Rachel Emma. "Yearly Reviews? Try Weekly." *Wall Street Journal*, September 6, 2011. <http://bit.ly/wsj-reviews>.

- Stone, Douglas, and Sheila Heen. *Thanks for the Feedback*. New York: Viking, 2014.

## 亲密性：从个人到团队

- Fowler, Chad. “Your Most Important Skill: Empathy.” ChadFowler.com, January 19, 2014. <http://bit.ly/fowler-empathy>.
- Granovetter, Mark S. “The Strength of Weak Ties.” *American Journal of Sociology* 78, no. 6 (May 1973).
- Herting, Stephen R. “Trust Correlated with Innovation Adoption in Hospital Organizations.” Paper presented for the American Society for Public Administration, National Conference, Phoenix, Arizona, March 8, 2002.
- Hewstone, Miles, Mark Rubin, and Hazel Willis. “Intergroup Bias.” *Annual Review of Psychology* 53 (2002).
- Hunt, Vivian, Dennis Layton, and Sara Prince. “Why Diversity Matters.” McKinsey.com, January 2015. <http://bit.ly/mckinsey-diversity>.
- Kohtamaki, Marko, Tauno Kekale, and Riitta Viitala. “Trust and Innovation: From Spin-Off Idea to Stock Exchange.” *Creativity and Innovation Management* 13, no. 2 (June 2004).
- Mind Tools Editorial Team. “The Greiner Curve: Understanding the Crises That Come with Growth.” MindTools.com, N.d. <http://bit.ly/greiner-curve>.
- Schwartz, Katrina. “How Do You Teach Empathy? Harvard Pilots Game Simulation.” KQED.org, May 9, 2013. <http://bit.ly/teach-empathy>.
- Sussna, Jeff. “Empathy: The Essence of devops.” Engineering.IT, January 11, 2014. <http://bit.ly/sussna-empathy>.

## 工具：文化加速器

- Allspaw, John. “A Mature Role for Automation: Part 1.” KitchenSoap.com, September 21, 2012. <http://bit.ly/allspaw-automation>.



- Caum, Carl. “Continuous Delivery vs. Continuous Deployment: What’s the Diff?” Puppet blog, August 30, 2013. <http://bit.ly/cd-vs-cd>.
- Coutinho, Rodrigo. “In Support of DevOps: Kanban vs. Scrum.” DevOps.com, July 29, 2014. <http://bit.ly/kanban-v-scrum>.
- Cowie, Jon. *Customizing Chef*. Sebastopol, CA: O’Reilly Media, 2014.
- Dixon, Jason. *Monitoring with Graphite*. Sebastopol, CA.: O’Reilly Media, 2015.
- Forsgren, Nicole, and Jez Humble. “The Role of Continuous Delivery in IT and Organizational Performance.” In the Proceedings of the Western Decision Sciences Institute (WDSI), Las Vegas, Nevada, October 27, 2015.
- Friedman, Ron. “Schedule a 15-Minute Break Before You Burn Out.” *Harvard Business Review*. August 4, 2014. <http://bit.ly/hbr-breaks>.
- Humble, Jez. “Deployment pipeline anti-patterns.” <http://bit.ly/humble-antipatterns>.
- Kim, Gene. Kanbans and DevOps: Resource Guide for *The Phoenix Project* (Part 2).” IT Revolution Press, N.d. <http://bit.ly/kanbans-devops>.
- Konnikova, Maria. “The Open-Office Trap.” *The New Yorker*, January 7, 2014. <http://bit.ly/open-office-trap>.
- Ōno, Taiichi. *Toyota Production System*. Cambridge, MA: Productivity Press, 1988.
- Rembetsy, Michael, and Patrick McDonnell. “Continuously Deploying Culture.” Etsy presentation at Velocity London 2012. <http://vimeo.com/51310058>.

## 规模化：拐点

- Clark, William. “Explores Motivation Research—A Boss’ Tool.” *Chicago Tribune*, August 4, 1959.
- Cole, Jonathan R., and Stephen Cole. “The Ortega Hypothesis.” *Science* 178, no. 4059 (1972): 368–375.
- Engel David, Anita W. Woolley, Lisa X. Jing, Christopher F. Chabris, and Thomas

- W. Malone. "Reading the Mind in the Eyes or Reading Between the Lines? Theory of Mind Predicts Collective Intelligence Equally Well Online and Face-to-Face." *PLoS ONE* 9, no. 12 (2014).
- Grant, Adam M. *Give and Take*. New York: Viking, 2013.
  - Griswold, Alison. "Here's Why Eliminating Titles and Managers at Zappos Probably Won't Work." *Business Insider*, January 6, 2014. <http://bit.ly/holacracyunlikely>.
  - Hackman J. R. "The Design of Work Teams." In *The Handbook of Organizational Behavior*, edited by Jay W. Lorsch. Englewood Cliffs, NJ: Prentice-Hall, 1987.
  - Hackman, J. Richard, and Greg R. Oldham. "Motivation Through the Design of Work: Test of a Theory." *Organizational Behavior and Human Performance* 16, no. 2 (1976): 250–279.
  - Kurtz, Cynthia F., and David J. Snowden. "Bramble Bushes in a Thicket: Narrative and the Intangibles of Learning Networks." In *Strategic Networks: Learning to Compete*, edited by Michael Gibbert and Thomas Durand. Malden, MA: Blackwell, 2007.
  - Mickman, Heather, and Ross Clanton. "DevOps at Target." Posted on October 29, 2014. <http://bit.ly/devops-target>.
  - Puppet. "2015 State of DevOps Report." <http://bit.ly/2015-state-of-devops>.
  - Rose, Katie. "Performance Assessment with Impact." *devopsdays Silicon Valley* 2015. Posted on November 13, 2015. <http://bit.ly/rose-perf-assess>.
  - Shannon-Solomon, Rachel. "devops Is Great for Startups, but for Enterprises It Won't Work—Yet." *Wall Street Journal*, May 13, 2014. <http://bit.ly/wsj-devopsenterprise>.
  - Tanizaki, Jun'ichirō. *In Praise Of Shadows*. New Haven, CT: Leete's Island Books, 1977.

## 搭建devops文化桥梁

- Fox, Martha Lane. “Directgov 2010 and Beyond: Revolution not Evolution.” GOV.UK, November 23, 2010. <http://bit.ly/fox-directgov-2010>.
- Gillespie, Nicole A., and Leon Mann. “Transformational Leadership and Shared Values: The Building Blocks of Trust.” *Journal of Managerial Psychology* 19, no. 6 (2004).
- Indiana University. “Women in Mostly Male Workplaces Exhibit Psychological Stress Response.” EurekAlert, August 24, 2015. <http://bit.ly/women-maleworkplace>.
- Reed, J. Paul. *DevOps in Practice*. Sebastopol, CA: O'Reilly Media, 2013.

## 推荐会议和线下聚会

- !!Con (<http://bangbangcon.com>)
- AlterConf (<http://www.alterconf.com>)
- Berlin Buzzwords (<https://berlinbuzzwords.de>)
- CoffeeOps (<http://www.coffeeops.org>)
- CSSconf EU (<http://www.cssconf.eu>)
- devopsdays (<http://www.devopsdays.org>)
- Infracoders (<http://infrastructurecoders.com>)
- JSConf EU (<http://www.jsconf.eu>)
- Open Source and Feelings (<http://osfeels.com>)
- Open Source Bridge (<http://opensourcebridge.org>)
- Monitorama (<http://monitorama.com>)
- SassConf (<http://sassconf.com>)
- Strange Loop (<http://www.thestrangeloop.com>)



- Velocity (<http://conferences.oreilly.com/velocity>)
- XoXo (<http://www.xoxofest.com>)

## 推荐播客

- Arrested DevOps (<https://www.arresteddevops.com>)
- DevOps Cafe Podcast with John Willis and Damon Edwards (<http://devopscafe.org>)
- Food Fight Show (<http://foodfightshow.org>)

## 作者介绍

---

**Jennifer Davis**是devopsdays的全球组织者，也是硅谷devopsdays的当地组织者，她还是Coffeeops的创办人。她支持了旧金山地区的多个社区线下聚会。在Chef任职期间，Jennifer 开发了Chef cookbooks来简化构建和管理基础设施。她曾在多个有关devops、技术文化、监控和自动化的行业会议上发表演讲。不工作时，她喜欢在湾区的山间小径漫步，学着做东西，和她的爱人Brian和她的狗George共渡美好时光。

**Ryn Daniels**是一位高级运维工程师，在Etsy任职。她热爱自动化和运维，并把这种热爱转化为监控、配置管理和运维工具开发方面的专业才能，她曾在众多行业会议上发表演讲，包括Velocity、devopsdays和Monitorama，主题涵盖基础设施自动化、规模化监控解决方案和工程中的文化改变。Ryn是devopsdays NYC的合作组织者之一，并帮助管理Ladies Who Linux in New York。她与很大一群猫住在布鲁克林，在她的闲暇时间里，她喜欢演奏大提琴、攀岩还有喝啤酒。

## 封面介绍

---

本书的封面动物是一头野牦牛(*Bos mutus*)。这是一种看上去很可怕但很友好的牛科动物，主要生活在遥远的青藏高原西北部的山区，这也是哺乳动物最高的栖息地。

野牦牛的突出特点包括高高隆起的肩部以及几乎垂到地面的长长的黑毛。这是牛科体型最大的成员之一，除了这种野牦牛，其他大型牛科动物还包括美洲野牛、非洲水牛和家牛。公牛肩部高度在5到7英尺之间，体重可以达到2200磅；母牛的体型通常只有公牛的三分之一。

野牦牛有很大的肺活量、很高的红细胞数而且还有一身温暖的毛外衣，所以能很好地适应它的高海拔栖息地。尽管它们块头很大，野牦牛却是攀登能手，它们可以利用分瓣的蹄子和强壮的牛腿在岩石覆盖的冰原上穿行。它们有坚硬的牛角，从它们的大脑袋两侧向外弯曲，使它们可以在冰雪中寻找食物。反过来，它们对温暖的气候非常敏感，会根据季节迁徙躲避高温天气。

野牦牛是群居、温顺的食草动物，以草、草本植物和地衣为食。母牛和它们的小牛在一起生活，这种牛群可以有多达100头牛；公牛往往更喜欢独处，尽管也会成群四处活动，但公牛群通常只有10头牛左右。他们会结伴长途跋涉找寻植物。

尽管人们认为全世界的野牦牛数超过1200万头，但这个数字主要包括个头较小的驯养牦牛。野生牦牛被认为是易危物种，在过去30年间，据报道它们的数量下降了30%以上。野牦牛最大的威胁是非法狩猎，不过与家养牦牛的杂交也是导致纯种野牦牛数量下降的一个因素。野外的野牦牛平均寿命为23年。

O'Reilly图书封面上的很多动物都濒临灭绝，所有这些动物对我们的世界都很重要。要了解如何帮助这些动物，请访问[animals.oreilly.com](http://animals.oreilly.com)。

封面图片来自Lydekker's Royal Natural History。



# Effective DevOps (中文版)

有些公司认为，采用devops就意味着需要引入专家或者大量新工具。利用这本实用指南，你会了解为什么devops是一个专业的文化运动，它要求从你的组织内部开始改变。本书作者提供了多种方法来改善团队内的协作、创建团队之间的亲密性，促进公司高效地使用工具，以及在组织拐点过程中完成规模化工作。

devops强调迭代的工作，从而打破信息孤岛、监督关系，以及修正组织中团队之间以及团队内部产生的误解。通过应用这本书中的实战策略，不论在你的组织中处于哪个层次，你都可以在你的环境中完成可持续的改变。

- 探讨devops的基础，并了解有效实现devops的4大支柱。
- 鼓励协作，帮助个人协同工作并建立持久而长期的关系。
- 建立团队之间的亲密性，同时平衡不同的目标或指标。
- 通过选择有利于组织的工具和工作流，加快文化转变。
- 排查组织整个生命周期中可能出现的常见问题和误区。
- 学习组织和个人的案例，帮助完成你自己的devops之旅。

**Jennifer Davis**是Chef的一位软件工程师，着力开发开源软件来简化基础设施的构建和管理。她创办了Coffeeops来帮助人们通过协作、合作和饮料建立社区。

**Ryn Daniels**是Etsy的一位高级运维工程师，主要关注监控、配置管理和运维工具开发。她帮助组织了devopsdays和Ladies Who Linux in New York。

“本书对人为因素做了广泛、深入的调查，每一个希望建立高绩效技术团队和组织的管理者都应当仔细研究这些因素。”

——Jez Humble

*Continuous Delivery* (Addison-Wesley)  
和*Lean Enterprise* (O'Reilly)  
的合作者

“本书是一本全面优秀的技术合集，这些构成了出版《敏捷宣言》以来技术工作的最大变革。”

——Mandi Walls

Chef的技术社区经理和  
*Building a devops Culture*  
(O'Reilly) 作者

“通过强调人以及追求卓越所需的交互，这本书会让组织中每个层次的每一个人都有收获。”

——Gene Kim

*Phoenix Project:  
A Novel About IT, DevOps,  
and Helping Your Business Win*  
(IT Revolution Press) 的合作者

IT MANAGEMENT/CULTURE

O'Reilly Media, Inc. 授权中国电力出版社出版

此简体中文版仅限于在中华人民共和国境内（但不允许在中国香港、澳门特别行政区和中国台湾地区）销售发行  
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

ISBN 978-7-5198-1419-9



定价：88.00元